MS_DSP_458 Artificial Intelligence and Deep Learning

A.4 Fourth Research/Programming Assignment (Deep Learning)

Rajeev Sharma

Northwestern University

Course - MS_DSP

December 3, 2023

A.4 Fourth Research/Programming Assignment

(1) Abstract. An executive summary of the research:

- In this assignment we are concerned with deep learning, a neural network with more than one hidden layer. We want to create deep neural networks and to analyze how various factors affect the fitting and ultimate test set performance of these networks. We think in practical terms: "How can we build trustworthy deep learning networks?"
- Here we think in terms of exploring alternative network structures/topologies, such as convolutional neural networks with differing structures in terms of convolution and pooling layers.
- Among the network topologies (structures) we test in this assignment, there is a network that employs convolution operators (that is, a convolutional neural network). We also compare processing requirements and predictive accuracy of networks with two, three, or more hidden layers, with at least one of those layers involving a convolution operator.
- Within each network topology, we evaluate alternative settings for hyperparameters.
- For each network under study, we also check out the possible contribution of dropout as a regularization technique, useful for addressing issues with variance.
- Objective is to get experimental results by classifying five fungi types using different deep learning approaches like convolutional neural network models - VGG16, VGG19, ResNet50, Efficient Net, EfficientNetV2 along with simpler deep neural networks (with fewer hidden layers and fewer nodes per layer) in order to benchmark the classification performance using pre-trained models based on the ImageNet dataset. We also study effect of different hyperparameters on accuracy of the concerned deep neural networks.

(2) Introduction:

Traditionally, diagnosis and treatment of fungal infections in humans depend heavily on face-to-face consultations or examinations made by specialized laboratory scientists known as mycologists. In many cases, such as the recent mucormycosis spread in the COVID-19 pandemic, an initial treatment can be safely suggested to the patient during the earliest stage of the mycological diagnostic process by performing a direct examination of biopsies or samples through a microscope.

To classify five fungi types using different convolutional neural network models like VGG16, VGG19, and ResNet50, the images donated by a mycological laboratory in Colombia have been used in this project. These images were manually labelled into five classes and curated with subject matter expert assistance. The images were later cropped and modified with automated coding routines to produce the final dataset. The dataset being used for this project is available at: https://drive.google.com/drive/folders/1z98N6bomZluImU0VXsfJl_spphAjZt9o?usp=sharing.

Benefits:

Computer-aided diagnosis systems using deep learning models have been trained and used for the late mycological diagnostic stages. However, there are no reference literature works for the early stages. Results of this study can be used as initial point of reference as mentioned in research article - Defungi: Direct mycological examination of microscopic fungi images (https://doi.org/10.58190/icat.2023.12) to encourage future research works to improve classification performance over the target classes and enhance the methodology used by present Computer-aided diagnosis systems.

(3) Literature review:

Following research papers mention research on Fungi image classification that will serve as reference for this project:

- Research article - Defungi: Direct mycological examination of microscopic fungi images (https://doi.org/10.58190/icat.2023.12) provides a comprehensive review of experimental results classifying five fungi types.
- Classification of fungal genera from microscopic images using artificial intelligence (https://doi.org/10.1016/j.jpi.2023.100314) provides classification of pathogenic fungi from microscopic images using deep convolutional neural networks (CNN).

(4) Methods:

The research in this assignment focuses on diagnosing early mycological stages by using Computer-aided diagnosis systems using deep learning models like VGG16, VGG19, ResNet50, Efficient Net, EfficientNetV2 along with simpler deep neural networks (with fewer hidden layers and fewer nodes per layer) trained on above mentioned dataset. We plan to train our classification models using convolutional neural network models. We target test accuracy above 80% for this project as we have limited training data. For this project, I have used libraries such as TensorFlow or PyTorch for building and training the CNN, and libraries like Matplotlib or Seaborn for data visualization. We have utilized accelerator of Kaggle GPU - T4 * 2 and T4-GPU in Google Colab for running the codes in attached jupyter notebooks.

a. Review research design and modelling methods:

Fungal infections, due to their diverse manifestations and varying characteristics, present significant challenges in medical diagnosis. This research delves into applying deep-learning techniques for detecting fungal infections from microscopic fungal images. By harnessing the power of Convolutional Neural Networks (CNNs), we propose an approach that employs transfer learning to accurately classify different fungal species. We also train nine CNN models (Model 3, Model 4, Model 7, Model 8, Model 9, Model 10, Model 11, Model 12 and Model 13) trained from scratch on Defungi dataset. This dataset comprises microscopic images of five fungal types. We aim to boost performance by fine-tuning/training the models' dense(top) layers. The results achieved underscore the effectiveness of our deep learning

approach in precisely identifying and classifying fungal infections. This holds promising potential to aid medical professionals in timely and accurate diagnoses. The findings presented in this research contribute to ongoing research in medical image analysis and drive advancements in the field of automated disease detection.

This research gives us hands-on, practical experience with not only designing, training, and assessing a neural network for its accuracy in image classification, and interpreting the impact of hyperparameters, but goes one step further. Regarding exploration, the goal of this research is to understand how the neurons/nodes in hidden layers in a neural network have learned to represent features within the input data. It gives us hands-on, practical experience with understanding the transition from simple (single hidden layer) to deep (multiple hidden layers) networks.

This hinges on understanding how hidden nodes learn to extract features from their inputs. When there are multiple hidden node layers, each successive layer extracts more generalized and abstract features.

When a hidden layer "learns" the kinds of features that are inherent in its input data, it is using a generative method. In this case, we're not telling it what those feature classes are; it has to figure them out on its own.

b. Implementation and programming:

The first approach benchmarks the classification performance for nine CNN models (Model 3, Model 4, Model 7, Model 8, Model 9, Model 10, Model 11, Model 12 and Model 13) trained from scratch on Defungi dataset. Second approach benchmarks the classification performance on Defungi dataset using pre-trained models (VGG16, VGG19, ResNet50, Efficient Net, EfficientNetV2) based on the weights learnt from ImageNet dataset.

The classes included in the Defungi dataset are superficial fungi compatible with dermatophytes and moulds. Their details are mentioned below for reference:

Class		**Fungi Type**
Hypha 1	->	Tortuous septate hyaline hyphae
Hypha 2	->	Beaded arthroconidial septate hyaline hyphae
Hypha 3	->	Groups or mosaics of arthroconidia
Hypha 5	->	Septate hyaline hyphae with chlamydioconidia
Hypha 6	->	Broad brown hyphae

Hypha is represented by 'H' in the labels in jupyter notebooks attached for your reference. So, the fungi classes become: H1, H2, H3, H5 and H6.

c. Data preparation, exploration, visualization:

We load defungi dataset which has 7290 images in training data set and 1824 images in test data set. Total images in dataset are 9114. Visualisation of these classes is given in Figure 1 of Appendix-A at end of this report. Detailed information of Defungi dataset is given in Figure 2 in Appendix-A. We split this dataset into train and test sets with split-ratio of 20% for test dataset.

(5) Results:

Results review along with model evaluation is as follows:

1. **Model 3** (Pls. see Figure 3 in Appendix for the model):

(Pls. note: Here 3 is only indicative in model name with no consequence regarding sequence or number of models. Same applies to other models in this study.)

In this model we create CNN with 2 convolutional layers/max pooling layers (no regularization). First convolution layer has 256 filters with kernel size (3,3) and strides (1,1) and relu activation. Second convolution layer has 512 filters with kernel size (3,3) and strides (1,1) and relu activation. Both convolution layers are followed by max pooling layers. After second max pooling layer, we flatten the output and then pass it to dense layer with 512 neurons and activation: relu. Output layer has 5 neurons and has softmax activation. We get 10627589 trainable parameters. We compile this model with optimizer: adam, loss: CategoricalCrossentropy and metrics as accuracy. Then we fit the model on train data with epochs=20, batch size=512 with callback = Earlystopping by monitoring val_accuracy with patience = 2. While using this method we also use early stopping with patience equal to 2. This means that once the model fails to improve valuation accuracy for 2 consecutive epochs, further training will stop. Our test accuracy is 59.714% which can certainly be improved. Train loss drops sharply while validation loss increases and then

decreases and again increases. Train accuracy increases in steps but validation accuracy first decreases and then increases to value more than train accuracy and then drops again sharply.

2. **Model 4** (Pls. see Figure 4 in Appendix for the model):

Now we create CNN with 3 convolutional layers/max pooling layers (no regularization). First convolution layer has 256 filters with kernel size (3,3) and strides (1,1) and relu activation. Second convolution layer has 512 filters with kernel size (3,3) and strides (1,1) and relu activation. Third convolution layer has 1024 filters with kernel size (3,3) and strides (1,1) and relu activation. All convolution layers are followed by max pooling layers. After third max pooling layer, we flatten the output and then pass it to dense layer with 512 neurons and activation: relu. Output layer has 5 neurons and has softmax activation. We get 8007173 trainable parameters. We perform model4 compiling and fitting just like for model3. Our test accuracy is 59.22% which can certainly be improved. Train loss drops sharply while validation loss decreases and again increases. Train accuracy increases in steps but validation accuracy first increases and then decreases sharply to value much lower than train accuracy and then rises again sharply.

Now we redo models above with dropout regularization technique.

3. **Model 7** (Pls. see Figure 5 in Appendix for the model):

Model 7 is Model 3 with dropout layer added. We perform model7 compiling and fitting just like for model3. Our test accuracy is 61.31 % which can certainly be improved. Train loss drops sharply while validation loss increases and again decreases. Train and validation accuracy increase in steps but validation accuracy is more than train accuracy. This shows that our model 7 is not over fitting. This shows that drop-out layer has regularisation effect.

4. **Model 8** (Pls. see Figure 6 in Appendix for the model):

Model 8 is Model 4 with dropout layer added. We perform model8 compiling and fitting just like for model4. Our test accuracy is 59.714 % which can certainly be improved. Train loss drops sharply while validation loss also decreases at a slower rate. Train and validation accuracy increase in steps but validation accuracy is more than train accuracy. This shows that our model 8 is not over fitting. This shows that drop-out layer has regularisation effect.

5. **Model 9** (Pls. see Figure 7 in Appendix for the model):

Model 9 is Model 7 with change in kernel size from (3,3) to (5,5). We perform model9 compiling and fitting just like for model7. Our test accuracy is 49.12 % which can certainly be improved. Train loss drops sharply while validation loss also decreases at same rate. Train and validation accuracy increase in steps but validation accuracy is more than train accuracy. This shows that our model 9 is not over fitting. This shows that increasing kernel size has no marginal increase in accuracy in test accuracy.

6. **Model 10** (Pls. see Figure 8 in Appendix for the model):

Model 10 is Model 7 with change in kernel stride from (1,1) to (2,2). We perform model10 compiling and fitting just like for model7. Our test accuracy is 59.824 % which can certainly be improved. Train loss drops sharply while validation loss also decreases at similar rate. Train and validation accuracy increase in steps but validation accuracy is more than train accuracy. This shows that our model 10 is not over fitting. This shows that increasing kernel stride has reduced test accuracy.

Model 11 (Pls. see Figure 9 in Appendix for the model):

We use model_7 and add an extra Conv2d layer before each pooling step to see how that performs and this gives us model 11. Our test accuracy is 53.56 % which can certainly be improved. Train loss drops sharply while validation loss also decreases at similar rate. Train

and validation accuracy increase in steps but validation accuracy is more than train accuracy. This shows that our model 11 is not over fitting.

7. **Model 12** (Pls. see Figure 10 in Appendix for the model):

In model 12, we'll start with simply allowing model 7 to run for more iterations (epochs=25) to see if that increases the test accuracy. Our test accuracy is 48.79 % which can certainly be improved. Train loss drops sharply while validation loss increases and then stabilises at same level as train loss. Train and validation accuracy first decrease and then increase in steps finally becoming constant but validation accuracy is more than train accuracy. This shows that our model 12 is not over fitting. Increasing number of epochs does not increase test accuracy for model 12 as model 7 has test accuracy of 61.31%.

8. **Model 13** (Pls. see Figure 11 in Appendix for the model):

In model 13, we'll start with simply allowing model 11 to run for more iterations (epochs=25) to see if that increases the test accuracy. Our test accuracy is 48.79 % which can certainly be improved. Train loss drops sharply while validation loss increases and then stabilises at same level as train loss. Train and validation accuracy first decrease and then become constant but validation accuracy is more than train accuracy. This shows that our model 13 is not over fitting. Increasing number of epochs does not increase test accuracy for model 13 as model 11 has test accuracy of 53.56% while model 13 has test accuracy of 48.79%.

9. **VGG19 (pretrained model):** (Pls. see Figure 12 in Appendix for the model):

Here, we have used VGG19 pretrained model that was trained on the ImageNet data. We have pulled this directly from Keras and use the trained weights in the lower layers while still adding and training our own dense upper layers (just like we have on previous models). But we train it for only 20 epochs due to computing resource constraint. Here we get test accuracy of 89.18% and this is the highest till now. Train loss drops sharply while validation loss decreases and then stabilises at a lower level as compared to train loss. Train and validation accuracy increase and then become constant but validation accuracy is more than train

accuracy. This shows that our model **VGG19 (pretrained model)** is not over fitting. Confusion matrix in Appendix shows that most of the classes are correctly predicted.

10. **VGG16 (pretrained model):** (Pls. see Figure 13 in Appendix for the model): Here, we have used VGG16 pretrained model that was trained on the ImageNet data. We have pulled this directly from Keras and use the trained weights in the lower layers while still adding and training our own dense upper layers (just like we have on previous models). But we train it for only 20 epochs due to computing resource constraint. Here we get test accuracy of 88.47% and this is second highest till now. Train loss drops sharply while validation loss decreases and then stabilises at a lower level as compared to train loss. Train and validation accuracy increase with number of epochs but validation accuracy is more than train accuracy. This shows that our model **VGG16 (pretrained model)** is not over fitting. Confusion matrix in Appendix shows that most of the classes are correctly predicted.

11. **EfficientNetB0 (pretrained model):** (Pls. see Figure 14 in Appendix for the model): Here, we have used EfficientNetB**0** pretrained model that was trained on the ImageNet data. We have pulled this directly from Keras and used the trained weights in the lower layers while still adding and training our own dense upper layers (just like we have on previous models). But we train it for only 20 epochs due to computing resource constraint. Here we get test accuracy of 84.79% and this is thirs highest till now. Train loss drops sharply while validation loss decreases at a slower rate and then stabilises at a higher level as compared to train loss. Train and validation accuracy increase with number of epochs but validation accuracy is less than train accuracy. This shows that our model **EfficientNetB0 (pretrained model)** is appropriately fitting.

12. **ResNet50 (pretrained model):** (Pls. see Figure 15 in Appendix for the model): Here, we have used ResNet50 pretrained model that was trained on the ImageNet data. We have pulled this directly from Keras and used the trained weights in the lower layers while still adding and training our own dense upper layers (just like we have on previous models). But we train it for only 20 epochs due to computing resource constraint. Here we get test accuracy of 84.15% and this is fourth highest till now. Train loss drops sharply while validation loss decreases and then stabilises at a higher level as compared to train loss. Train and validation accuracy increase with number of epochs but validation accuracy is lesser than train accuracy. This shows that our model **ResNet50 (pretrained model)** is appropriately fitting.

13. **EfficientNetV2B0 (pre-trained model) :** (Pls. see Figure 16 in Appendix for the model): Here, we have used **EfficientNetV2B0** pretrained model that was trained on the ImageNet data. We have pulled this directly from Keras and used the trained weights in the lower layers while still adding and training our own dense upper layers (just like we have on previous models). But we train it for only 20 epochs due to computing resource constraint. Here we get test accuracy of 81.33% and this is fifth highest till now. Train loss drops sharply while validation loss decreases and then stabilises at a higher level as compared to train loss. Train and validation accuracy increase with number of epochs but validation accuracy is lesser than train accuracy. This shows that our model **EfficientNetV2B0 (pretrained model)** is appropriately fitting.

Result1: Table containing the accuracy and loss for train/test/validation for ALL the models. **Pls. find table attached in Appendix as Result 1 for your reference**.

Jupyter notebooks for above models are also attached herewith for your reference.

Result2: For VGG19 model, VGG16 model and Model_3, we have extracted features from first 16 layers in VGG19, features from last max pooling layer of VGG16 and have extracted features from first four layers of Model-3. Pls. find fungi image and extracted features for these models: VGG19, VGG16 and Model-3 attached herewith in Appendix under Result 2 for your reference. Jupyter notebook for the same: Extracting-features-vgg19-vgg16-model3-layers-sharmarajeev_revised_file.ipynb is also attached herewith for your reference. We can see that features extracted by VGG19 layers are very much in detail as compared to those extracted by other models like Model_3. This shows why it ranks first in classification accuracy in models chosen for this study.

(6) Conclusions:

We would recommend to Management that we should go ahead with VGG19 model along with transfer learning using 'ImageNet' weights which has maximum test accuracy of 89.18%. and minimum test loss of 1.11. We will also hold 1822 samples (20% samples) from given Defungi dataset set as validation set during model fitting for 20 epochs and we will use callback with earlystopping while monitoring valuation accuracy with patience value of 2.

In conclusion, this experiment has shown that transfer learning, when combined with a powerful deep learning architecture like VGG19, can be effectively used for multi-task classification tasks. This approach could potentially be applied to a wide range of computer vision tasks, opening up new possibilities for the field. Looking ahead, this experiment can be expanded to other fungi, enabling doctors to easily distinguish between different types of fungi. This will essentially assist them in faster diagnosis and hence help provide more effective treatment for fungal infections.

We can also plan to create a user-friendly mobile application that will display the classification results for a broader variety of fungi. Furthermore, we can explore the impact of

various parameters, such as the activation function, optimization of the pooling function, and the loss function for enhancing classification accuracy of VGG19 model.

(7) Challenges:

However, it's imperative to acknowledge certain challenges encountered during this study. The limited size of the dataset, despite meticulous data augmentation efforts, poses a constraint on the model's ability to generalize to diverse fungi images. Future studies could benefit from an expanded and more diverse dataset, enabling the model to learn a broader range of features and nuances associated with different types of fungi. The interpretability of deep learning models, especially in medical diagnosis, remains a challenge. While VGG19 and VGG16 models exhibit high accuracy, efforts toward interpretability, explain ability, and transparency in decision-making should be prioritized to build trust in the model's predictions within the medical community.

Appendix

References:

Jupyter notebooks for 14 models and 1 jupyter notebook for feature extraction are as follows:

1. model3-sharma-rajeev.ipynb
2. model4-sharma-rajeev.ipynb
3. model7-sharma-rajeev.ipynb
4. model8-sharma-rajeev.ipynb
5. model9-sharma-rajeev.ipynb
6. model10-sharma-rajeev.ipynb
7. model11-sharma-rajeev.ipynb
8. model12-sharma-rajeev.ipynb
9. model13-sharma-rajeev.ipynb
10. vgg19_images_classification_final_Sharma_Rajeev.ipynb
11. vgg16_images_classification_final_Sharma_Rajeev.ipynb
12. defungi_classification_transfer_learning_resnet50_final.ipynb
13. EfficientNetB0_final_Sharma_Rajeev.ipynb
14. EfficientNetV2_images_classification_final_Sharma_Rajeev.ipynb
15. Extracting-features-vgg19-vgg16-model3-layers-sharmarajeev_revised_file.ipynb

Figure 1: Visualisation of fungi classes

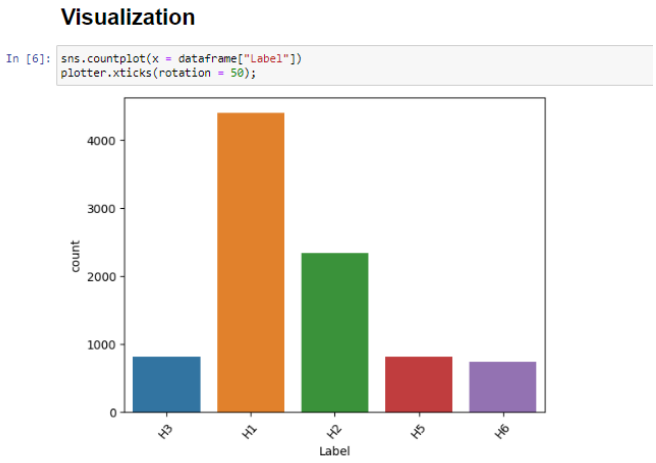


Figure 2: Detailed information of Defungi dataset

DETAILED INFORMATION ABOUT THE DATASET

Category	Number of Images
H1	4404
H2	2334
H3	819
H5	818
H6	739
Total	**9114**

Figure 3: Model 3

```
In [33]: model_3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 30, 30, 256)       7168

 max_pooling2d_4 (MaxPoolin  (None, 15, 15, 256)       0
 g2D)

 conv2d_5 (Conv2D)           (None, 13, 13, 512)       1180160

 max_pooling2d_5 (MaxPoolin  (None, 6, 6, 512)         0
 g2D)

 flatten_2 (Flatten)         (None, 18432)             0

 dense_4 (Dense)             (None, 512)               9437696

 dense_5 (Dense)             (None, 5)                 2565

=================================================================
Total params: 10627589 (40.54 MB)
Trainable params: 10627589 (40.54 MB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 4: Model 4

In [49]:
```
model_4 = models.Sequential()
model_4.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu, input_shape=(32, 32, 3)))
model_4.add(layers.MaxPooling2D((2, 2),strides=2))
model_4.add(layers.Conv2D(filters=512, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu))
model_4.add(layers.MaxPooling2D(pool_size=(2, 2),strides=2))
model_4.add(layers.Conv2D(filters=1024, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu))
model_4.add(layers.MaxPooling2D(pool_size=(2, 2),strides=2))
model_4.add(layers.Flatten())
model_4.add(layers.Dense(units=512, activation=tf.nn.relu))
model_4.add(layers.Dense(units=5, activation=tf.nn.softmax))
```

In [50]:
```
model_4.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 30, 30, 256)       7168

 max_pooling2d_9 (MaxPoolin  (None, 15, 15, 256)       0
 g2D)

 conv2d_10 (Conv2D)          (None, 13, 13, 512)       1180160

 max_pooling2d_10 (MaxPooli  (None, 6, 6, 512)         0
 ng2D)

 conv2d_11 (Conv2D)          (None, 4, 4, 1024)        4719616

 max_pooling2d_11 (MaxPooli  (None, 2, 2, 1024)        0
 ng2D)

 flatten_4 (Flatten)         (None, 4096)              0

 dense_8 (Dense)             (None, 512)               2097664

 dense_9 (Dense)             (None, 5)                 2565

=================================================================
Total params: 8007173 (30.54 MB)
Trainable params: 8007173 (30.54 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

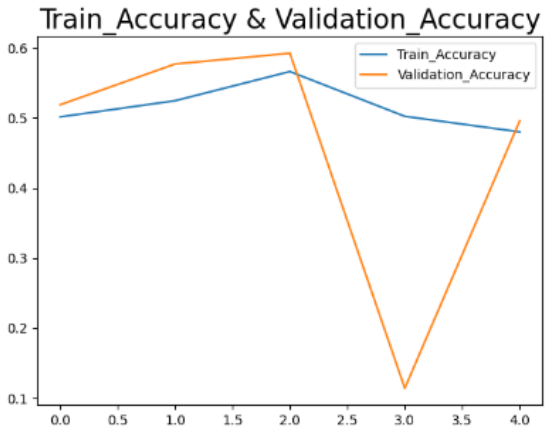


Figure 5: Model 7

In [59]:
```
model_7 = models.Sequential()
model_7.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu, input_shape=(32, 32, 3)))
model_7.add(layers.MaxPooling2D((2, 2),strides=2))
model_7.add(layers.Conv2D(filters=512, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu))
model_7.add(layers.MaxPooling2D(pool_size=(2, 2),strides=2))
model_7.add(layers.Flatten())
model_7.add(layers.Dropout(0.5))
model_7.add(layers.Dense(units=512, activation=tf.nn.relu))
model_7.add(layers.Dense(units=5, activation=tf.nn.softmax))
```

In [60]:
```
model_7.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 30, 30, 256)       7168

 max_pooling2d_12 (MaxPooli  (None, 15, 15, 256)       0
 ng2D)

 conv2d_13 (Conv2D)          (None, 13, 13, 512)       1180160

 max_pooling2d_13 (MaxPooli  (None, 6, 6, 512)         0
 ng2D)

 flatten_5 (Flatten)         (None, 18432)             0

 dropout (Dropout)           (None, 18432)             0

 dense_10 (Dense)            (None, 512)               9437696

 dense_11 (Dense)            (None, 5)                 2565

=================================================================
Total params: 10627589 (40.54 MB)
Trainable params: 10627589 (40.54 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

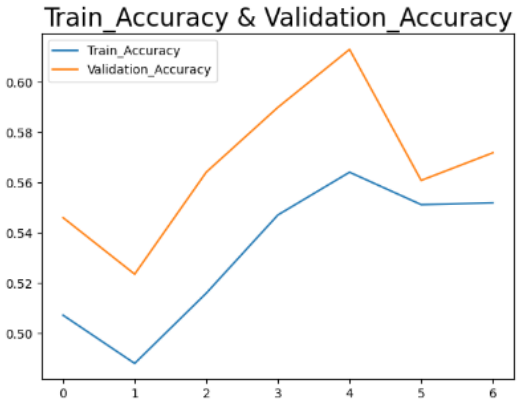
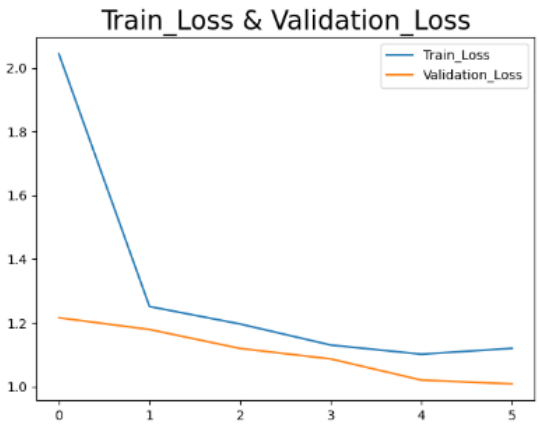


Figure 6: Model 8

```
In [69]: model_8 = models.Sequential()
model_8.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu, input_shape=(32, 32, 3)))
model_8.add(layers.MaxPooling2D((2, 2),strides=2))
model_8.add(layers.Conv2D(filters=512, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu))
model_8.add(layers.MaxPooling2D(pool_size=(2, 2),strides=2))
model_8.add(layers.Conv2D(filters=1024, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu))
model_8.add(layers.MaxPooling2D(pool_size=(2, 2),strides=2))
model_8.add(layers.Flatten())
model_8.add(layers.Dropout(0.5))
model_8.add(layers.Dense(units=512, activation=tf.nn.relu))
model_8.add(layers.Dense(units=5, activation=tf.nn.softmax))
```

```
In [70]: model_8.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_14 (Conv2D)          (None, 30, 30, 256)       7168

 max_pooling2d_14 (MaxPooli  (None, 15, 15, 256)       0
 ng2D)

 conv2d_15 (Conv2D)          (None, 13, 13, 512)       1180160

 max_pooling2d_15 (MaxPooli  (None, 6, 6, 512)         0
 ng2D)

 conv2d_16 (Conv2D)          (None, 4, 4, 1024)        4719616

 max_pooling2d_16 (MaxPooli  (None, 2, 2, 1024)        0
 ng2D)

 flatten_6 (Flatten)         (None, 4096)              0

 dropout_1 (Dropout)         (None, 4096)              0

 dense_12 (Dense)            (None, 512)               2097664

 dense_13 (Dense)            (None, 5)                 2565

=================================================================
Total params: 8007173 (30.54 MB)
Trainable params: 8007173 (30.54 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



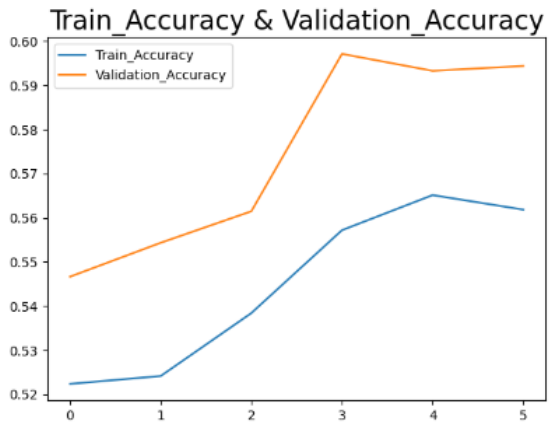
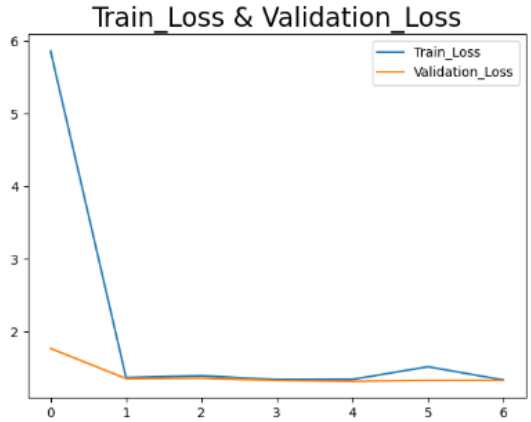


Figure 7: Model 9

In [79]:
```
model_9 = models.Sequential()
model_9.add(layers.Conv2D(filters=256, kernel_size=(5, 5), strides=(1, 1), activation=tf.nn.relu, input_shape=(32, 32, 3)))
model_9.add(layers.MaxPooling2D((2, 2),strides=2))
model_9.add(layers.Conv2D(filters=512, kernel_size=(5, 5), strides=(1, 1), activation=tf.nn.relu))
model_9.add(layers.MaxPooling2D(pool_size=(2, 2),strides=2))
model_9.add(layers.Flatten())
model_9.add(layers.Dropout(0.5))
model_9.add(layers.Dense(units=512, activation=tf.nn.relu))
model_9.add(layers.Dense(units=5, activation=tf.nn.softmax))
```

In [80]:
```
model_9.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_17 (Conv2D)          (None, 28, 28, 256)       19456

 max_pooling2d_17 (MaxPooli  (None, 14, 14, 256)       0
 ng2D)

 conv2d_18 (Conv2D)          (None, 10, 10, 512)       3277312

 max_pooling2d_18 (MaxPooli  (None, 5, 5, 512)         0
 ng2D)

 flatten_7 (Flatten)         (None, 12800)             0

 dropout_2 (Dropout)         (None, 12800)             0

 dense_14 (Dense)            (None, 512)               6554112

 dense_15 (Dense)            (None, 5)                 2565

=================================================================
Total params: 9853445 (37.59 MB)
Trainable params: 9853445 (37.59 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



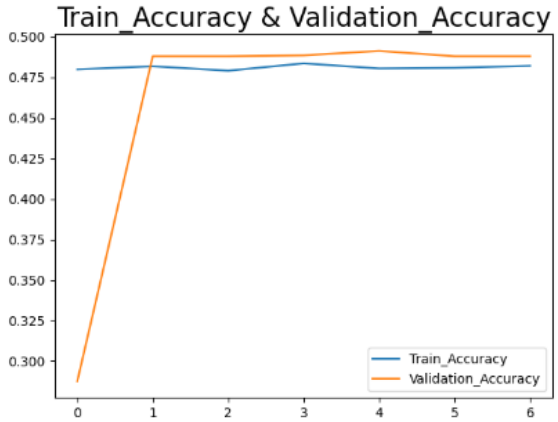
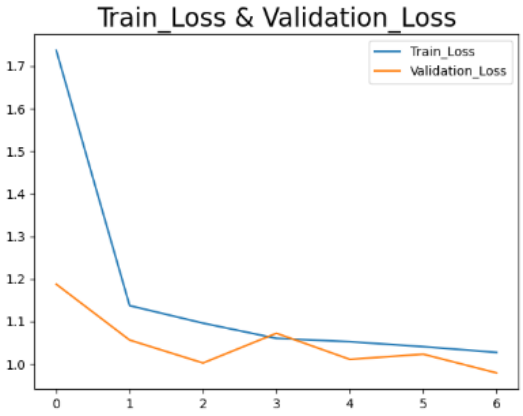


Figure 8: Model 10

In [89]:
```
model_10 = models.Sequential()
model_10.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides=(2, 2), activation=tf.nn.relu, input_shape=(32, 32, 3)))
model_10.add(layers.MaxPooling2D((2, 2),strides=2))
model_10.add(layers.Conv2D(filters=512, kernel_size=(3, 3), strides=(2, 2), activation=tf.nn.relu))
model_10.add(layers.MaxPooling2D(pool_size=(2, 2),strides=2))
model_10.add(layers.Flatten())
model_10.add(layers.Dropout(0.5))
model_10.add(layers.Dense(units=512, activation=tf.nn.relu))
model_10.add(layers.Dense(units=5, activation=tf.nn.softmax))
```

In [90]:
```
model_10.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_19 (Conv2D)          (None, 15, 15, 256)       7168

 max_pooling2d_19 (MaxPooli  (None, 7, 7, 256)         0
 ng2D)

 conv2d_20 (Conv2D)          (None, 3, 3, 512)         1180160

 max_pooling2d_20 (MaxPooli  (None, 1, 1, 512)         0
 ng2D)

 flatten_8 (Flatten)         (None, 512)               0

 dropout_3 (Dropout)         (None, 512)               0

 dense_16 (Dense)            (None, 512)               262656

 dense_17 (Dense)            (None, 5)                 2565

=================================================================
Total params: 1452549 (5.54 MB)
Trainable params: 1452549 (5.54 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



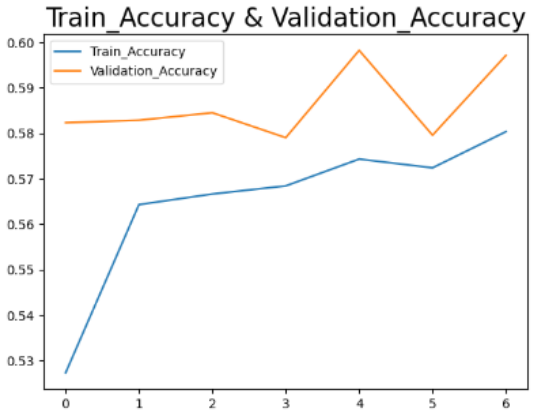


Figure 9: Model 11

```
In [99]: model_11 = models.Sequential()
model_11.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu, input_shape=(32, 32, 3)))
model_11.add(layers.Conv2D(filters=512, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu))
model_11.add(layers.MaxPooling2D((2, 2),strides=2))
model_11.add(layers.Conv2D(filters=512, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu))
model_11.add(layers.Conv2D(filters=512, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu))
model_11.add(layers.MaxPooling2D(pool_size=(2, 2),strides=2))
model_11.add(layers.Flatten())
model_11.add(layers.Dropout(0.5))
model_11.add(layers.Dense(units=512, activation=tf.nn.relu))
model_11.add(layers.Dense(units=5, activation=tf.nn.softmax))
```

```
In [100]: model_11.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_21 (Conv2D)          (None, 30, 30, 256)       7168

 conv2d_22 (Conv2D)          (None, 28, 28, 512)       1180160

 max_pooling2d_21 (MaxPooli  (None, 14, 14, 512)       0
 ng2D)

 conv2d_23 (Conv2D)          (None, 12, 12, 512)       2359808

 conv2d_24 (Conv2D)          (None, 10, 10, 512)       2359808

 max_pooling2d_22 (MaxPooli  (None, 5, 5, 512)         0
 ng2D)

 flatten_9 (Flatten)         (None, 12800)             0

 dropout_4 (Dropout)         (None, 12800)             0

 dense_18 (Dense)            (None, 512)               6554112

 dense_19 (Dense)            (None, 5)                 2565

=================================================================
Total params: 12463621 (47.54 MB)
Trainable params: 12463621 (47.54 MB)
Non-trainable params: 0 (0.00 Byte)
```

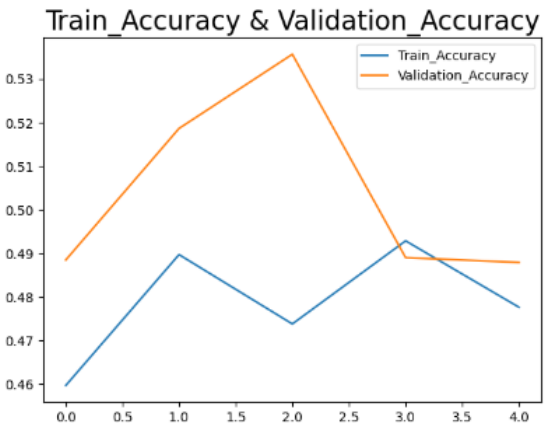
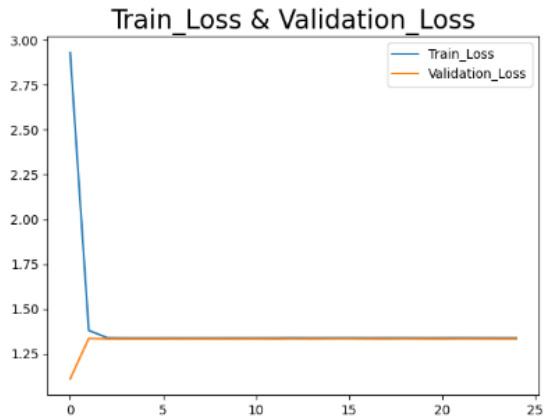


Figure 10: Model 12

```
In [109]: model_12 = models.Sequential()
model_12.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu, input_shape=(32, 32, 3)))
model_12.add(layers.MaxPooling2D((2, 2),strides=2))
model_12.add(layers.Conv2D(filters=512, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu))
model_12.add(layers.MaxPooling2D(pool_size=(2, 2),strides=2))
model_12.add(layers.Flatten())
model_12.add(layers.Dropout(0.5))
model_12.add(layers.Dense(units=512, activation=tf.nn.relu))
model_12.add(layers.Dense(units=5, activation=tf.nn.softmax))
```

```
In [110]: model_12.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_25 (Conv2D)          (None, 30, 30, 256)       7168

 max_pooling2d_23 (MaxPooli  (None, 15, 15, 256)       0
 ng2D)

 conv2d_26 (Conv2D)          (None, 13, 13, 512)       1180160

 max_pooling2d_24 (MaxPooli  (None, 6, 6, 512)         0
 ng2D)

 flatten_10 (Flatten)        (None, 18432)             0

 dropout_5 (Dropout)         (None, 18432)             0

 dense_20 (Dense)            (None, 512)               9437696

 dense_21 (Dense)            (None, 5)                 2565

=================================================================
Total params: 10627589 (40.54 MB)
Trainable params: 10627589 (40.54 MB)
Non-trainable params: 0 (0.00 Byte)
```



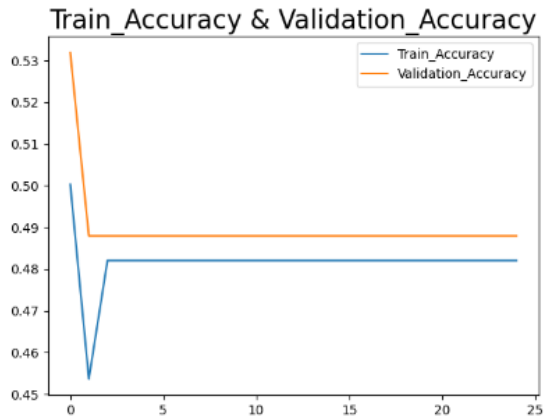
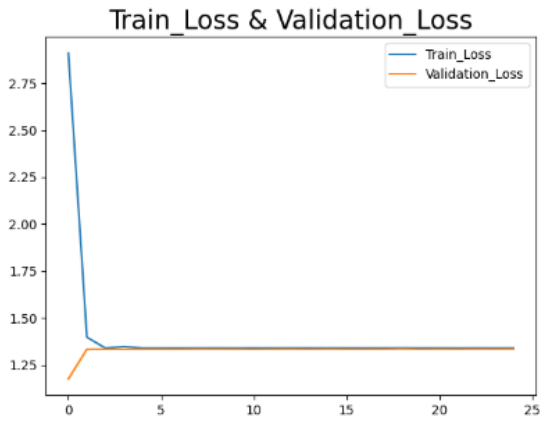


Figure 11: Model 13

In [119]:
```
model_13 = models.Sequential()
model_13.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu, input_shape=(32, 32, 3)))
model_13.add(layers.Conv2D(filters=512, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu))
model_13.add(layers.MaxPooling2D((2, 2),strides=2))
model_13.add(layers.Conv2D(filters=512, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu))
model_13.add(layers.Conv2D(filters=512, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu))
model_13.add(layers.MaxPooling2D(pool_size=(2, 2),strides=2))
model_13.add(layers.Flatten())
model_13.add(layers.Dropout(0.5))
model_13.add(layers.Dense(units=512, activation=tf.nn.relu))
model_13.add(layers.Dense(units=5, activation=tf.nn.softmax))
```

In [120]:
```
model_13.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_27 (Conv2D)          (None, 30, 30, 256)       7168

 conv2d_28 (Conv2D)          (None, 28, 28, 512)       1180160

 max_pooling2d_25 (MaxPooli  (None, 14, 14, 512)       0
 ng2D)

 conv2d_29 (Conv2D)          (None, 12, 12, 512)       2359808

 conv2d_30 (Conv2D)          (None, 10, 10, 512)       2359808

 max_pooling2d_26 (MaxPooli  (None, 5, 5, 512)         0
 ng2D)

 flatten_11 (Flatten)        (None, 12800)             0

 dropout_6 (Dropout)         (None, 12800)             0

 dense_22 (Dense)            (None, 512)               6554112

 dense_23 (Dense)            (None, 5)                 2565

=================================================================
Total params: 12463621 (47.54 MB)
Trainable params: 12463621 (47.54 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



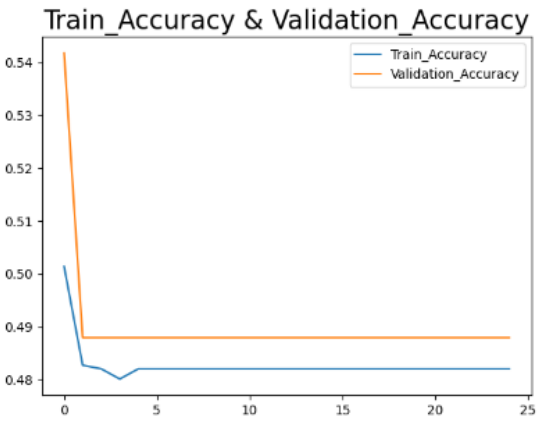
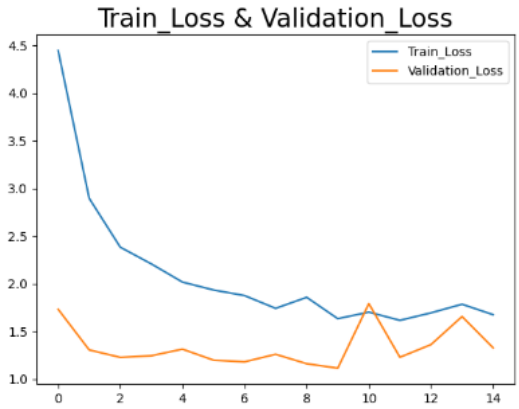
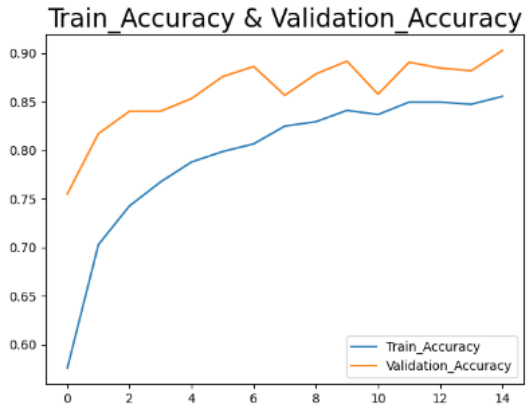


Figure 12: VGG19 pretrained model

```
In [10]: base_model = tf.keras.applications.VGG19(input_shape=(224,224,3),include_top=False,weights='imagenet')
base_model.trainable = False
keras_model=keras.models.Sequential()
keras_model.add(base_model)
keras_model.add(keras.layers.Flatten())
keras_model.add(keras.layers.Dropout(0.5))
keras_model.add(keras.layers.Dense(5,activation=tf.nn.softmax))
keras_model.summary()
```

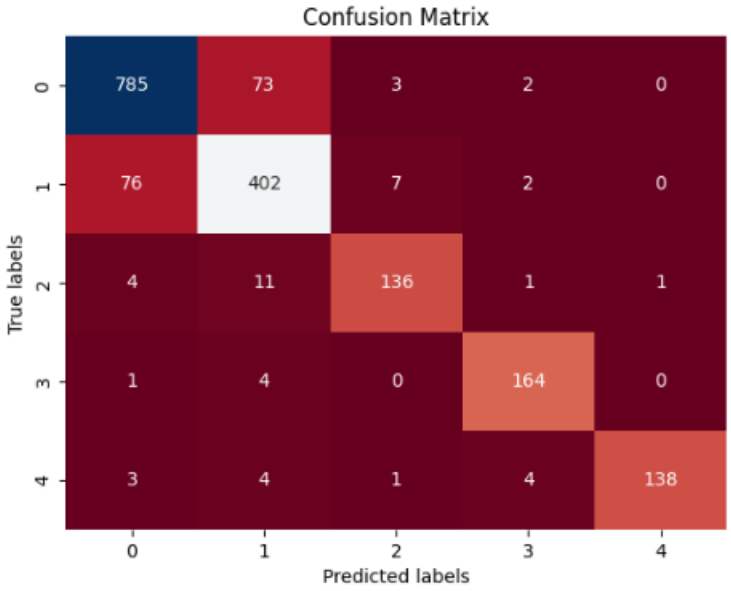
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 [==============================] - 1s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 7, 7, 512)         20024384

 flatten (Flatten)           (None, 25088)             0

 dropout (Dropout)           (None, 25088)             0

 dense (Dense)               (None, 5)                 125445

=================================================================
Total params: 20149829 (76.87 MB)
Trainable params: 125445 (490.02 KB)
Non-trainable params: 20024384 (76.39 MB)
_________________________________________________________________
```





Confusion_matrix

In [20]:
```
ax= plt.subplot()
CM = confusion_matrix(y_val,y_pred)
sns.heatmap(CM, annot=True, fmt='g', ax=ax,cbar=False,cmap='RdBu')
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
plt.show()
CM
```



Out[20]:
```
array([[785,  73,   3,   2,   0],
       [ 76, 402,   7,   2,   0],
       [  4,  11, 136,   1,   1],
       [  1,   4,   0, 164,   0],
       [  3,   4,   1,   4, 138]])
```

```
In [19]: plt.figure(figsize=(25,25))
         for i in range(32):
             ax = plt.subplot(8, 4, i + 1)
             plt.imshow(X_val[i].astype("uint8"))
             plt.title(f'{class_names[y_val[i]]} :: {class_names[y_pred[i]]}')
             plt.axis("off")
```

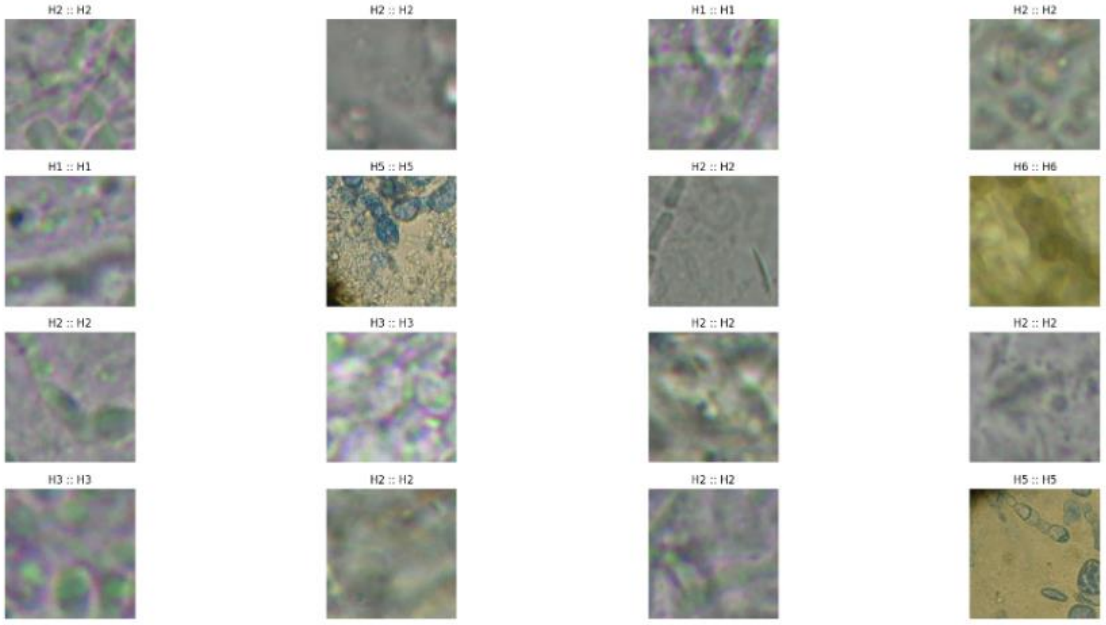
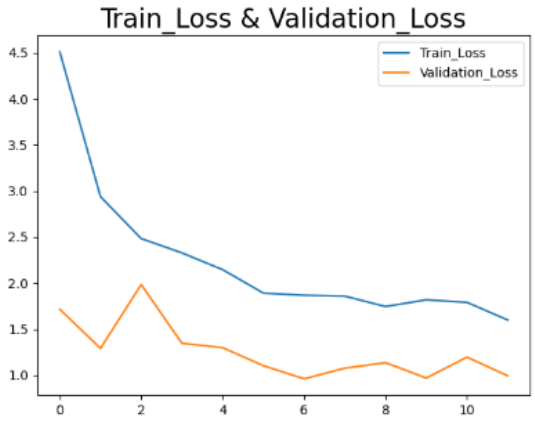
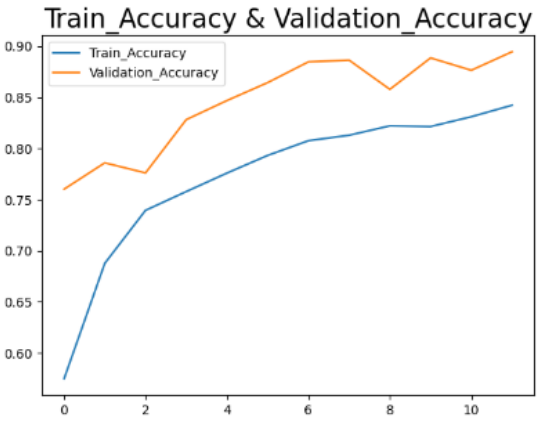


Figure 13: VGG16 pretrained model

```
In [22]: base_model = tf.keras.applications.VGG16(input_shape=(224,224,3),include_top=False,weights='imagenet')
         base_model.trainable = False
         keras_model=keras.models.Sequential()
         keras_model.add(base_model)
         keras_model.add(keras.layers.Flatten())
         keras_model.add(keras.layers.Dropout(0.5))
         keras_model.add(keras.layers.Dense(5,activation=tf.nn.softmax))
         keras_model.summary()
```

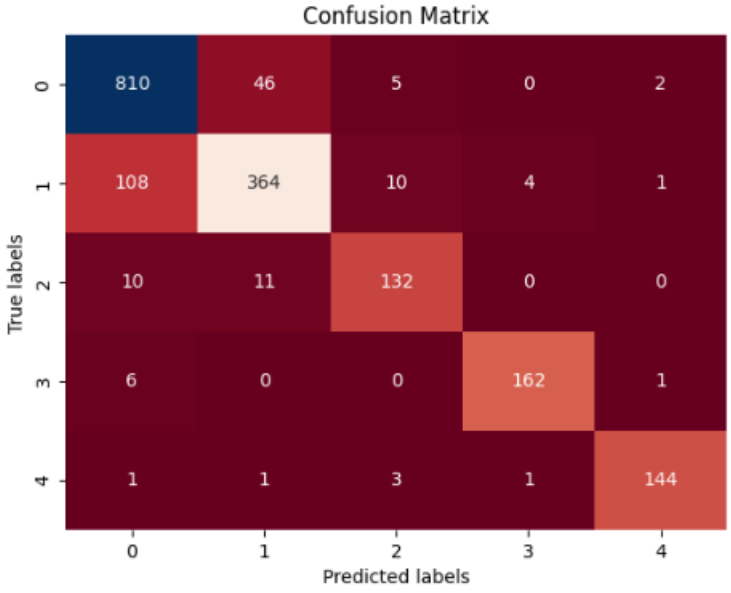
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten_1 (Flatten)         (None, 25088)             0

 dropout_1 (Dropout)         (None, 25088)             0

 dense_1 (Dense)             (None, 5)                 125445

=================================================================
Total params: 14840133 (56.61 MB)
Trainable params: 125445 (490.02 KB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```





Confusion_matrix

In [32]:
```
ax= plt.subplot()
CM = confusion_matrix(y_val,y_pred)
sns.heatmap(CM, annot=True, fmt='g', ax=ax,cbar=False,cmap='RdBu')
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
plt.show()
CM
```



Out[32]:
```
array([[810,  46,   5,   0,   2],
       [108, 364,  10,   4,   1],
       [ 10,  11, 132,   0,   0],
       [  6,   0,   0, 162,   1],
       [  1,   1,   3,   1, 144]])
```

Figure 14: EfficientNetB0 (pretrained model)

```
In [11]: base_model = tf.keras.applications.EfficientNetB0(
             input_shape=(224, 224, 3),
             include_top=False,
             weights='imagenet'
         )

Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
16705208/16705208 [==============================] - 0s 0us/step

In [12]: base_model.name

Out[12]: 'efficientnetb0'

In [13]: base_model.summary()

Model: "efficientnetb0"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0         []

 rescaling (Rescaling)       (None, 224, 224, 3)       0         ['input_1[0][0]']

 normalization (Normalizati  (None, 224, 224, 3)       7         ['rescaling[0][0]']
 on)

 rescaling_1 (Rescaling)     (None, 224, 224, 3)       0         ['normalization[0][0]']

 stem_conv_pad (ZeroPadding  (None, 225, 225, 3)       0         ['rescaling_1[0][0]']
 2D)

 stem_conv (Conv2D)          (None, 112, 112, 32)      864       ['stem_conv_pad[0][0]']

 stem_bn (BatchNormalizatio  (None, 112, 112, 32)      128       ['stem_conv[0][0]']
```

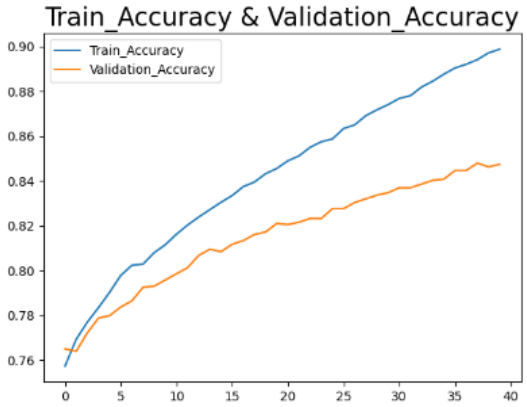


Figure 15: ResNet50 (pre-trained model)

5. Using Resnet50 As Base Model & Building the Model

In [8]:
```
resnet_model = ResNet50(include_top=False,weights="imagenet")
x = resnet_model.output
x = GlobalAveragePooling2D()(x)
x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(1024,activation='relu')(x)
x = tf.keras.layers.Dropout(0.6)(x)
x = tf.keras.layers.Dense(512,activation='relu')(x)
x = tf.keras.layers.Dropout(0.5)(x)
res = tf.keras.layers.Dense(5, activation="softmax")(x)

model = Model(inputs=resnet_model.input, outputs=res)

# Freezing the pre-trained layers so that we do not have to update weights ourselves
for layer in resnet_model.layers:
    layer.trainable = False

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 5s 0us/step
```

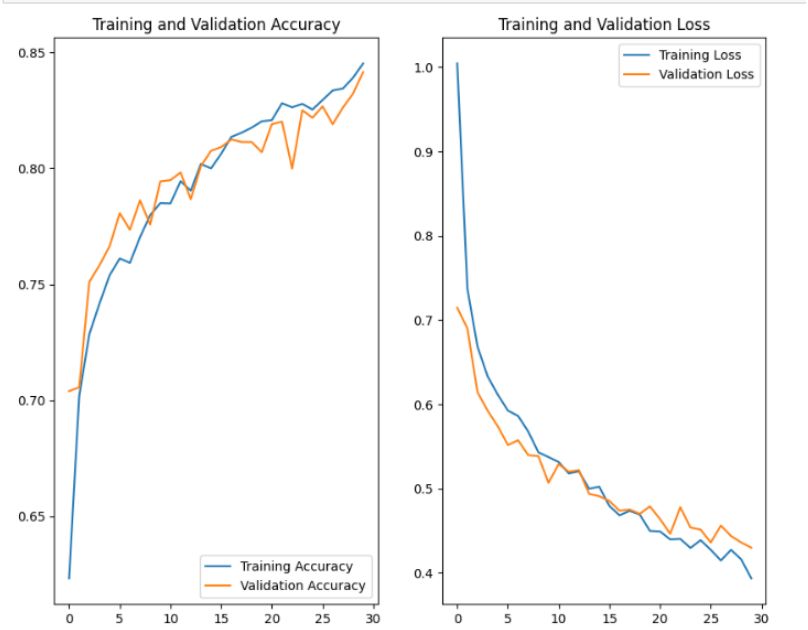
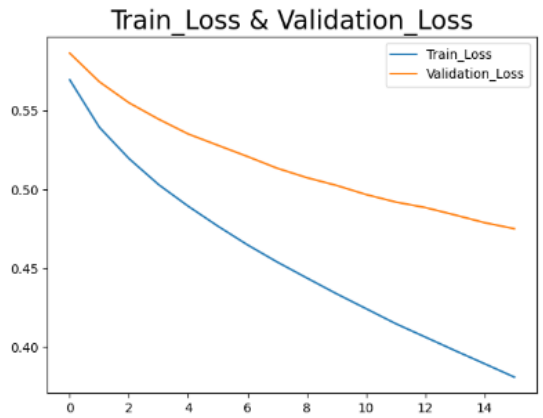
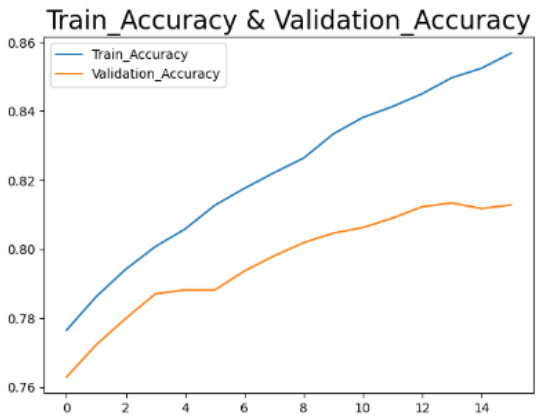


Figure 16: EfficientNetV2B0 (pre-trained model)

```
In [11]: base_model = tf.keras.applications.EfficientNetV2B0(
             input_shape=(224, 224, 3),
             include_top=False,
             weights='imagenet'
         )

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/efficientnet_v2/efficientnetv2-b0_notop.h5
24274472/24274472 [==============================] - 1s 0us/step

In [12]: base_model.name
Out[12]: 'efficientnetv2-b0'

In [13]: base_model.summary()
Model: "efficientnetv2-b0"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0         []

 rescaling (Rescaling)       (None, 224, 224, 3)       0         ['input_1[0][0]']

 normalization (Normalizati  (None, 224, 224, 3)       0         ['rescaling[0][0]']
 on)

 stem_conv (Conv2D)          (None, 112, 112, 32)      864       ['normalization[0][0]']

 stem_bn (BatchNormalizatio  (None, 112, 112, 32)      128       ['stem_conv[0][0]']
 n)

 stem_activation (Activatio  (None, 112, 112, 32)      0         ['stem_bn[0][0]']
 n)
```





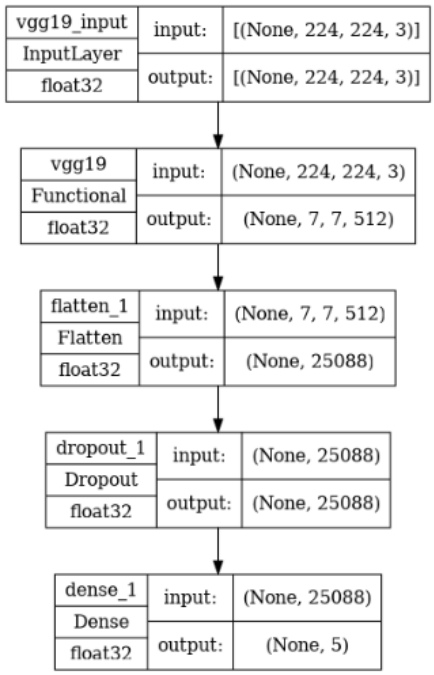
Result1: Performance of Models 1-14

Experiment	Model	Description	Train loss	Train accuracy	Test loss	Test accuracy
1	3	CNN with 2 conv. Layers	1.13	55.50%	1.03	59.71%
2	4	CNN with 3 conv. Layers	1.38	48.00%	1.015	59.22%
3	7	CNN with 2 conv. Layers and dropout	1.14	55.20%	1.05	61.30%
4	8	CNN with 3 conv. Layers and dropout	1.12	56.18%	1.086	59.71%
5	9	CNN with 2 conv. Layers and dropout with kernel size 5	1.33	48.20%	1.32	49.12%
6	10	CNN with 2 conv. Layers and dropout with stride 2	1.027	58.04%	1.01	59.82%
7	11	CNN with 2 conv. Layers and dropout and extra Conv2d layer before each pooling step	1.49	47.76%	1.12	53.56%
8	12	Running model_7 for 25 epochs	1.34	48.20%	1.33	48.79%
9	13	Running model_11 for 25 epochs	1.33	48.20%	1.34	48.80%
10	VGG19	Using pre-trained VGG19 model with dense layer	1.674	85.56%	1.11	89.18%
11	VGG16	Using pre-trained VGG16 model with dense layer	1.59	84.23%	0.96	88.47%
12	EfficientNetB0	Using pre-trained EfficientNetB0 model with dense layer	0.28	89.88%	0.42	84.79%
13	ResNet50	Using pre-trained ResNet50 model with dense layer	0.39	84.53%	0.43	84.15%
14	EfficientNetV2B0	Using pre-trained EfficientNetV2B0 model with dense layer	0.38	85.68%	0.483	81.33%

Result 2: For VGG19 model, VGG16 model and Model_3, we have extracted features from first 16 layers in VGG19, features from last max pooling layer of VGG16 and have extracted features from first four layers of Model-3. Pls. find fungi image and extracted features for these models: VGG19, VGG16 and Model-3 attached herewith for reference.

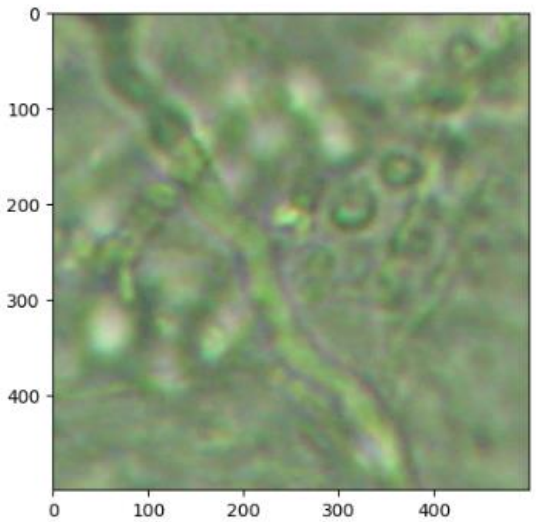
In [25]:
```
tf.keras.utils.plot_model(keras_model, to_file='model.png', show_shapes=True, show_layer_names=True,show_dtype=True,dpi=80)
```

Out[25]:



###Extracting features of below fungi image from layers of above VGG19 model

In [53]:
```
# Showing original fungus image for reference. We will extract features for this image from different
#layers of VGG19 model


from PIL import Image
import matplotlib.pyplot as plt

img_path = '/kaggle/input/defungi/H2/H2_51b_3.jpg'

# Read the image
img = Image.open(img_path)

# Display the image
plt.imshow(img)
plt.show()
```



##Extracting features of above fungi image from flatten_1 layer of above VGG19 model

In [18]:
```
model1 = Model(inputs=keras_model.input, outputs=keras_model.get_layer('flatten').output)

img_path = '/kaggle/input/defungi/H2/H2_51b_3.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

flatten_features = model1.predict(x)
```

```
1/1 [==============================] - 1s 800ms/step
```

In [19]:
```
flatten_features
```

Out[19]:
```
array([[8.028913, 0.      , 0.      , ..., 0.      , 0.      , 0.      ]],
      dtype=float32)
```

In [77]:
```
flatten_features.shape
```

Out[77]:
```
(1, 25088)
```

##Extracting features of above fungi image from dropout_1 layer of above VGG19 model

In [21]:
```
model1 = Model(inputs=keras_model.input, outputs=keras_model.get_layer('dropout').output)

img_path = '/kaggle/input/defungi/H2/H2_51b_3.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

dropout_1_features = model1.predict(x)
```

```
1/1 [==============================] - 0s 155ms/step
```

In [22]:
```
dropout_1_features
```

Out[22]:
```
array([[8.028913, 0.      , 0.      , ..., 0.      , 0.      , 0.      ]],
      dtype=float32)
```

In [76]:
```
dropout_1_features.shape
```

Out[76]:
```
(1, 25088)
```

##Extracting features of above fungi image from dense_1 layer of above VGG19 model

In [24]:
```
model1 = Model(inputs=keras_model.input, outputs=keras_model.get_layer('dense').output)

img_path = '/kaggle/input/defungi/H2/H2_51b_3.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

dense_1_features = model1.predict(x)
```

```
1/1 [==============================] - 0s 164ms/step
```

In [25]:
```
dense_1_features
```

Out[25]:
```
array([[1.6180717e-03, 5.4455331e-06, 9.9837363e-01, 2.5897212e-16,
        2.8688719e-06]], dtype=float32)
```

In [75]:
```
dense_1_features.shape
```

Out[75]:
```
(1, 5)
```

Extracting features of above fungi image from block4_pool layer of VGG19 model

In [26]:
```
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg19 import preprocess_input
from tensorflow.keras.models import Model
import numpy as np

base_model = VGG19(weights='imagenet')
model = Model(inputs=base_model.input, outputs=base_model.get_layer('block4_pool').output)

img_path = '/kaggle/input/defungi/H2/H2_51b_3.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

block4_pool_features = model.predict(x)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels.h5
574710816/574710816 [==============================] - 4s 0us/step
1/1 [==============================] - 0s 118ms/step
```

In [27]:
```
print(block4_pool_features)
```

```
[[[[0.0000000e+00 0.0000000e+00 0.0000000e+00 ... 0.0000000e+00
    1.0786523e+02 0.0000000e+00]
   [0.0000000e+00 0.0000000e+00 0.0000000e+00 ... 0.0000000e+00
    2.7484982e+01 0.0000000e+00]
   [0.0000000e+00 0.0000000e+00 0.0000000e+00 ... 0.0000000e+00
    0.0000000e+00 0.0000000e+00]
   ...
   [0.0000000e+00 0.0000000e+00 0.0000000e+00 ... 4.3617737e+02
    0.0000000e+00 0.0000000e+00]
   [0.0000000e+00 1.2813208e+02 0.0000000e+00 ... 0.0000000e+00
    0.0000000e+00 0.0000000e+00]
   [0.0000000e+00 0.0000000e+00 0.0000000e+00 ... 0.0000000e+00
    0.0000000e+00 0.0000000e+00]]

  [[0.0000000e+00 1.8389163e+02 1.4130353e+01 ... 1.0383007e+02
    1.5725993e+02 5.2239456e+01]
   [0.0000000e+00 0.0000000e+00 0.0000000e+00 ... 0.0000000e+00
    1.4054515e+02 0.0000000e+00]
   [0.0000000e+00 0.0000000e+00 0.0000000e+00 ... 0.0000000e+00
    0.0000000e+00 0.0000000e+00]
```

In [74]:
```
block4_pool_features.shape
```

Out[74]: (1, 14, 14, 512)

Extracting features of above fungi image after last max pool layer of VGG16 model

In [28]:
```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input
import numpy as np

model = VGG16(weights='imagenet', include_top=False)

img_path = '/kaggle/input/defungi/H2/H2_51b_3.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

features = model.predict(x)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
1/1 [==============================] - 0s 134ms/step
```

In [29]:
```
print(features)
```

```
[[[[ 0.         0.         0.        ...  0.         0.
     0.       ]
   [ 0.         0.         0.        ...  0.         0.
     0.       ]
   [ 0.         0.         0.        ...  0.         1.439362
     0.       ]
   ...
   [ 0.         0.         0.        ...  0.         0.
     0.       ]
   [ 0.         0.         0.        ...  0.         0.
     0.       ]
   [ 0.         0.         0.        ...  0.         0.
     0.       ]]

  [[ 0.         0.         0.        ...  0.         0.
     0.       ]
   [ 0.         0.         0.        ...  0.         0.
     0.       ]
   [ 0.         0.         0.        ...  0.         0.
     0.       ]
   ...
```

In [73]:
```
features.shape
```

Out[73]: (1, 7, 7, 512)

Extracting features of first fungi image from train data set from first 16 layers of VGG19 model

In [85]:
```
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg19 import preprocess_input
import numpy as np

model_11 = VGG19(weights='imagenet', include_top=False)

img_path = '/kaggle/input/defungi/H2/H2_51b_3.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

features_11 = model_11.predict(x)
```

```
1/1 [==============================] - 0s 151ms/step
```

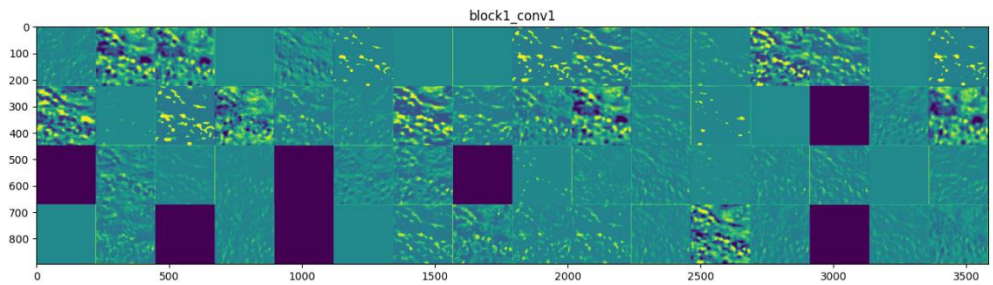
In [86]:
```
features_11.shape
```

Out[86]: (1, 7, 7, 512)

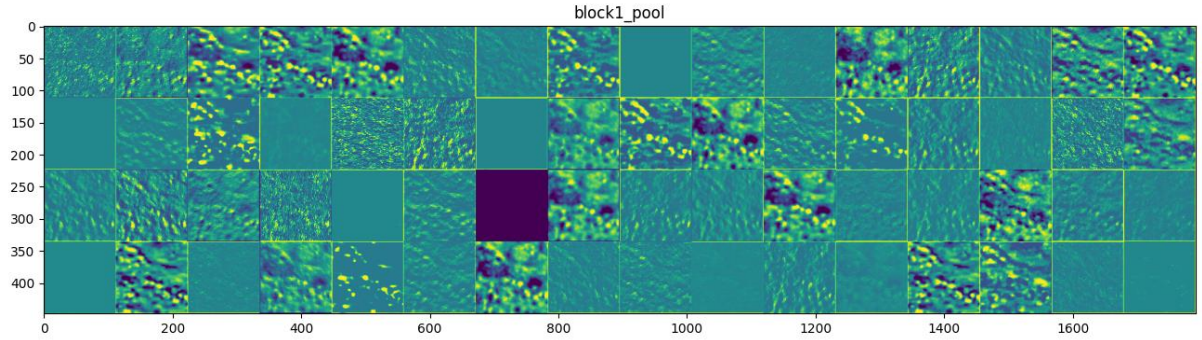
```
In [93]: #taking out first image from train dataset and prepping into the necessary shape
         first_tensor = next(iter(train.take(1)))[0]

In [94]: print(first_tensor.shape)

         (16, 224, 224, 3)

In [102]: # Extracts the outputs of the top  layers:
          layer_outputs = [layer.output for layer in model_11.layers[:16]]

In [103]: # Creates a model that will return these outputs, given the model input:
          activation_model = models.Model(inputs=model_11.input, outputs=layer_outputs)
          activations = activation_model.predict(first_tensor)

          1/1 [==============================] - 0s 140ms/step

In [104]: len(activations)

Out[104]: 16

In [105]: for i in range(len(activations)):
            print(activations[i].shape)

          (16, 224, 224, 3)
          (16, 224, 224, 64)
          (16, 224, 224, 64)
          (16, 112, 112, 64)
          (16, 112, 112, 128)
          (16, 112, 112, 128)
          (16, 56, 56, 128)
          (16, 56, 56, 256)
          (16, 56, 56, 256)
          (16, 56, 56, 256)
          (16, 56, 56, 256)
          (16, 28, 28, 256)
          (16, 28, 28, 512)
          (16, 28, 28, 512)
          (16, 28, 28, 512)
          (16, 28, 28, 512)
```

Some of the features extracted from first train image by first few layers of VGG19 are given below. Kindly refer attached notebook: Extracting-features-vgg19-vgg16-model3-layers-sharma-rajeev_revised_file.ipynb for complete details.

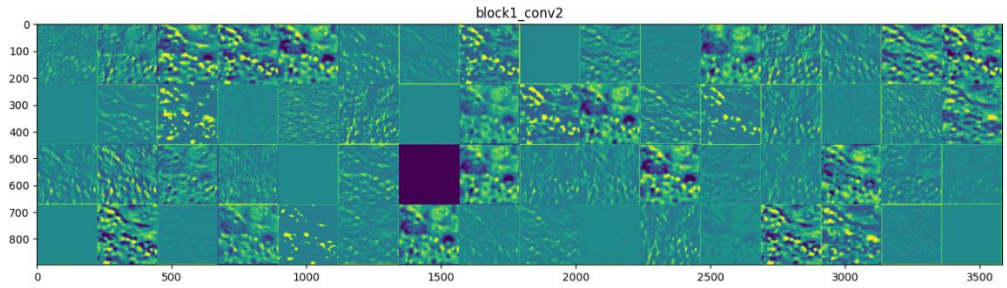
- **features extracted by block1_conv1 (first convolution block) are as follows:**



- **features extracted by block1_pool (first max pool block) are as follows:**

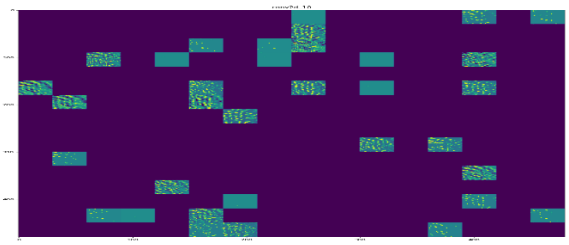


- **features extracted by block1_conv2 (second convolution block) are as follows:**

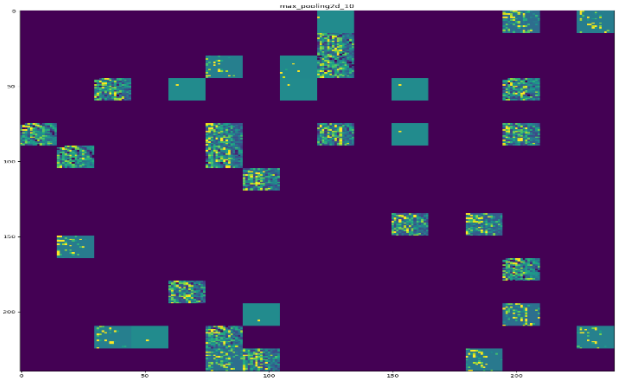


Some of the features extracted from first train image by first few layers of Model-3 are given below. Kindly refer attached notebook: Extracting-features-vgg19-vgg16-model3-layers-sharma-rajeev_revised_file.ipynb for complete details.

- **features extracted by conv2d_10 are as follows:**



- **features extracted by conv2d_10 are as follows:**



- **features extracted by conv2d_11 are as follows:**