This lab aims to make you more familiar with the implementation of some of the well-established deep classification models. You will learn to implement deep CNN models from scratch, compile, and train the models to achieve reproducible results. As you will be asked to make some comparisons, please save your observed results at each step.

Deeper Networks

Use the same framework of the previous lab (for loading the data and compiling the model) and replace the LeNet architecture by implementing the following figure (AlexNet architecture):

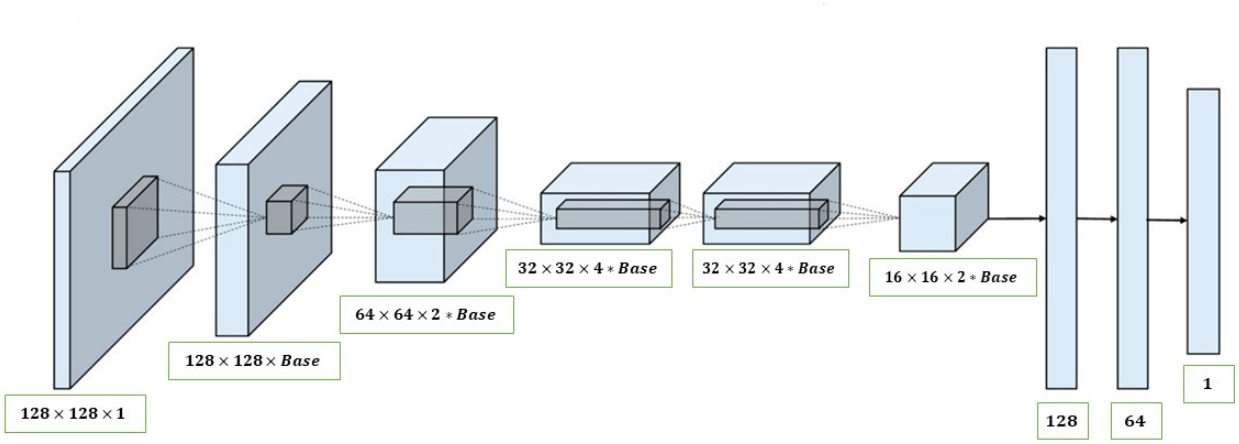

Fig.5 AlexNet Architecture.

```
def model(img_ch, img_width, img_height):

    model = Sequential()

    model.add(Conv2D(filters=n_base, input_shape=(img_width, img_height, img_ch),
                     kernel_size=(3,3), strides=(1,1), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Conv2D(filters= n_base *2, kernel_size=(3,3), strides=(1,1), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Conv2D(filters= n_base *4, kernel_size=(3,3), strides=(1,1), padding='same'))
    model.add(Activation('relu'))

    model.add(Conv2D(filters= n_base *4, kernel_size=(3,3), strides=(1,1), padding='same'))
    model.add(Activation('relu'))

    model.add(Conv2D(filters= n_base *2, kernel_size=(3,3), strides=(1,1), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Flatten())
    model.add(Dense(128))
    model.add(Activation('relu'))
```

```
    model.add(Dense(64))
    model.add(Activation('relu'))

    model.add(Dense(1))
    model.add(Activation('sigmoid'))

    model.summary()
    return model
```

Taks6A) Read the skin images with the size of 128*128 and train the AlexNet model with the following parameter: batch size = 8; epochs = 50; n_base(Base) = 32; learning rate = 0.0001, and Adam as optimizer. Evaluate the model performance.

Taks6B) Change the n_base parameter as 16 and 8 and run the model for 50 epochs. How do you interpret the observe results? Now, with n_base = 8, after each of the dense layer add a “drop out layer” with a drop out rate of 0.4 and train the model for 50 epochs. What is the effect of the drop out layer? Increase the number of epochs to 150. How do you explain the effect of increasing the number of epochs?

Task6C) Remove the drop out layers, set the parameters n_base=8, learning_rate = 1e-5 and run the model for n_epochs =150,350 epochs. How changing the learning rate parameter affect model performance?

Taks6D) For the fix parameters of learning_rate = 1e-5, n_base=8, n_epochs = 150, change the batch size parameters as 2,4,8. Do you see any significant differences in model performance?

Task6E) By finding the optimum values of batch_size, learning rate, and base parameters, train the model for 100 epochs and make sure it is not overfitted. Report the classification accuracy of the model. Then, for this model, only change the optimizer algorithm from Adam to SGD and RMSprop and compare the observed results.

Task6F) “Binary cross entropy (BCE)” is not the only loss function for a binary classification task. Run the code again by changing the loss func into “hinge” with Adam optimizer. What is the major difference between the “BCE” and “hinge” in terms of model performance? Please note, you need to have class labels as [-1, 1] therefore, you need to change the labels as :

```
y_test[y_test == 0] = -1
y_train[y_train == 0] = -1
```

Taks7A) The purpose of this task is to implement an architecture from scratch. You have to implement the architecture of the VGG16 model, as shown in the following figure. Please note 3 ×3 represents the size of the convolutional kernels, “Base” shows the number of feature maps, “pool/2” indicates the max-pooling operator, and “fc 64” means fully connected (dense) layer with 64 neurons.

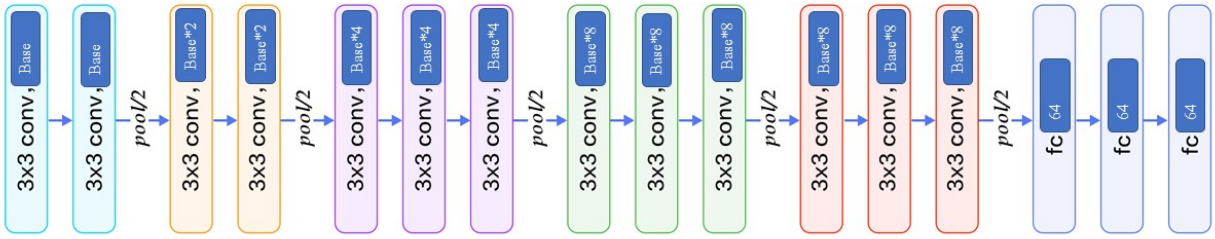

Fig.6 VGG16 Architecture.

Task7B) Load the Skin data set with the size of 128*128. For the following fixed parameters, compile the model and find the optimum value of the learning rate that will lead to reliable classification performance over the validation set. What is the proper learning rate?

n_epoch = 100, batch_s = 8, n_base = 8

Task7C) For the same setting as task6C (only for n_epoch=150) compare the classification accuracy over the validation set between the AlexNet and VGG16 models. How do you justify the observed results?

Task7D) Change the parameter n_base to 16. Train the model for 100 epochs with LR = 1e-5 and batch size of 8. Does increasing the number of feature maps affect model performance? Then, add a dropout layer with a rate of 0.2 for each of the dense layer. What was the effect of adding the dropout layer?

Task7E) So far, you classified the Skin cancer dataset with 3 different models named as LeNet, AlexNet as well VGG16. In general, what is the difference between these three models? Which of them yields more accurate classification results? Why? To evaluate the model performance, how do you assess the loss values? How can you prevent the model training from overfitting?

Task8) Train the VGG16 model developed in Task7 with the following parameters to classify two types of bone fractures from “Bone” dataset. Please note the size of the Bone images is quite large, so that it would take a longer time to read and load all the images.

```
data_path = '.../Lab1/Bone/'
img_w, img_h = 128, 128
replace the pattern1 and pattern2 in gen_label function with ‘AFF’ and ‘NFF’
Loss function: BCE, Optimizer: Adam, Learning Rate: ?, n_base: 8 & 16, n_epoch: 100, batch_s = 8
```

Task9) With the implemented VGG models, which of the Skin/Bone image sets classified more accurately? Why? How do you make sure that achieved results are reliable?

Task10) In previous exercises, you conducted some experiments with binary classification tasks by implementing three deep networks named as LeNet, AlexNet, VGG. In this task, the data set includes X-ray images of 9 different organs. Therefore, you are expected to extend the implemented models into a multi-class classification task. Modify the data loader to load the

images along with their class labels properly. Extend the LeNet and AlexNet models for multi-class classification tasks. Tune these two models by finding the optimum values of hyperparameters to get the best performance for each of the models and, then, compare the observed results between the two models.
Report the learning curves for both of the loss and accuracy values (for train and test data).
`Data path is: '.../Lab1/X_ray/'`

==========

Bonus Task) Although residual networks are among the powerful classification models, their applications are not restricted only to the image classification models. In fact, for many different architectures, it would be useful to replace the conventional convolutional layers with residual blocks. Implement the AlexNet architecture with residual blocks and classify the Skin images. Then, compare the results between the original AlexNet and the one with residual blocks for the same hyperparameter values (e.g., n_base, batch_s, n_epoch, ...).

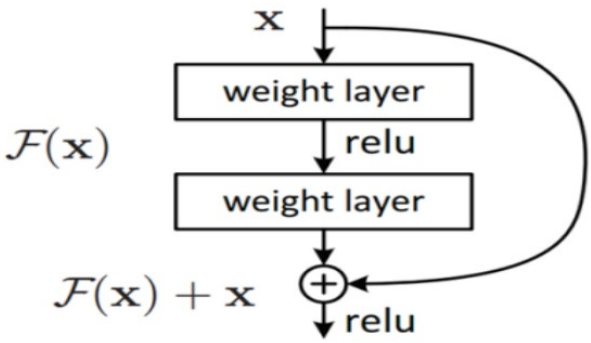

Fig.7 Schematic view of a residual block.

A simple yet effective implementation of residual blocks can be found at https://www.kaggle.com/meownoid/tiny-resnet-with-keras-99-314 .

Good luck.