Perform_Facial_Recognition_with_Deep_Learning_in_Keras_Using_CNN

Project: Perform Facial Recognition with Deep Learning in Keras Using CNN

Project Description

Problem Statement: Facial recognition is a biometric alternative that measures unique characteristics of a human face. Applications available today include flight check in, tagging friends and family members in photos, and "tailored" advertising. You are a computer vision engineer who needs to develop a face recognition programme with deep convolutional neural networks.

Objective: Use a deep convolutional neural network to perform facial recognition using Keras.

Dataset Details: ORL face database composed of 400 images of size 112 x 92. There are 40 people, 10 images per person. The images were taken at different times, lighting and facial expressions. The faces are in an upright position in frontal view, with a slight left-right rotation.

Step 1: Input the required libraries

[19]:
```python
# Data science libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import itertools

#Scikit-learn libraries
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve,auc

#Keras API Tensorflow 2 libraries
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D,␣
↪Activation, LeakyReLU
from keras.layers.noise import AlphaDropout
from keras.optimizers import Adam
```

```
from keras.utils.generic_utils import get_custom_objects
from keras import backend as K
from keras.callbacks import TensorBoard
from keras.utils.np_utils import to_categorical

print('Tensorflow version:', tf.__version__)
```

```
Tensorflow version: 2.1.0
```

[20]:
```
df = np.load('ORL_faces.npz') #loading dataset
```

Step 2: Load the dataset and preprocess the data

[21]:
```
# Loading train and test dataset (data is already split into)
x_train = df['trainX']
y_train = df['trainY']
x_test = df['testX']
y_test = df['testY']
```

[22]:
```
# Normalizing each image as each image is between 0-255 pixels
x_train = x_train.astype(np.float32) / 255.0
x_test = x_test.astype(np.float32) / 255.0

print('Training dataset shape: ',x_train.shape)
print('Testing dataset shape: ',x_test.shape)
```

```
Training dataset shape:  (240, 10304)
Testing dataset shape:  (160, 10304)
```

Step 3: Split the dataset

Split is done from Xtrain dataset into x_train and x_valid dataset

Here we considered only 10 % of the training dataset as validation dataset as number of images overall is very low (240)

[23]:
```
x_train, x_valid, y_train, y_valid =␣
 ↪train_test_split(x_train,y_train,test_size=0.1,random_state=42)
```

Step 4: Transform the images to equal sizes to feed in CNN

When we feed images in CNN the size of each image must be same.

- We will define the shape of image in terms of rows, columns
- To make equal size of all images (train, test, and valid dataset), we will use Reshape function

[24]:
```
# Shape of image definition
rows = 112
columns = 92
image_shape = (rows,columns,1)
```

[25]:
```
# Reshape function
x_train = x_train.reshape(x_train.shape[0],*image_shape)
x_test = x_test.reshape(x_test.shape[0],*image_shape)
x_valid = x_valid.reshape(x_valid.shape[0],*image_shape)
```

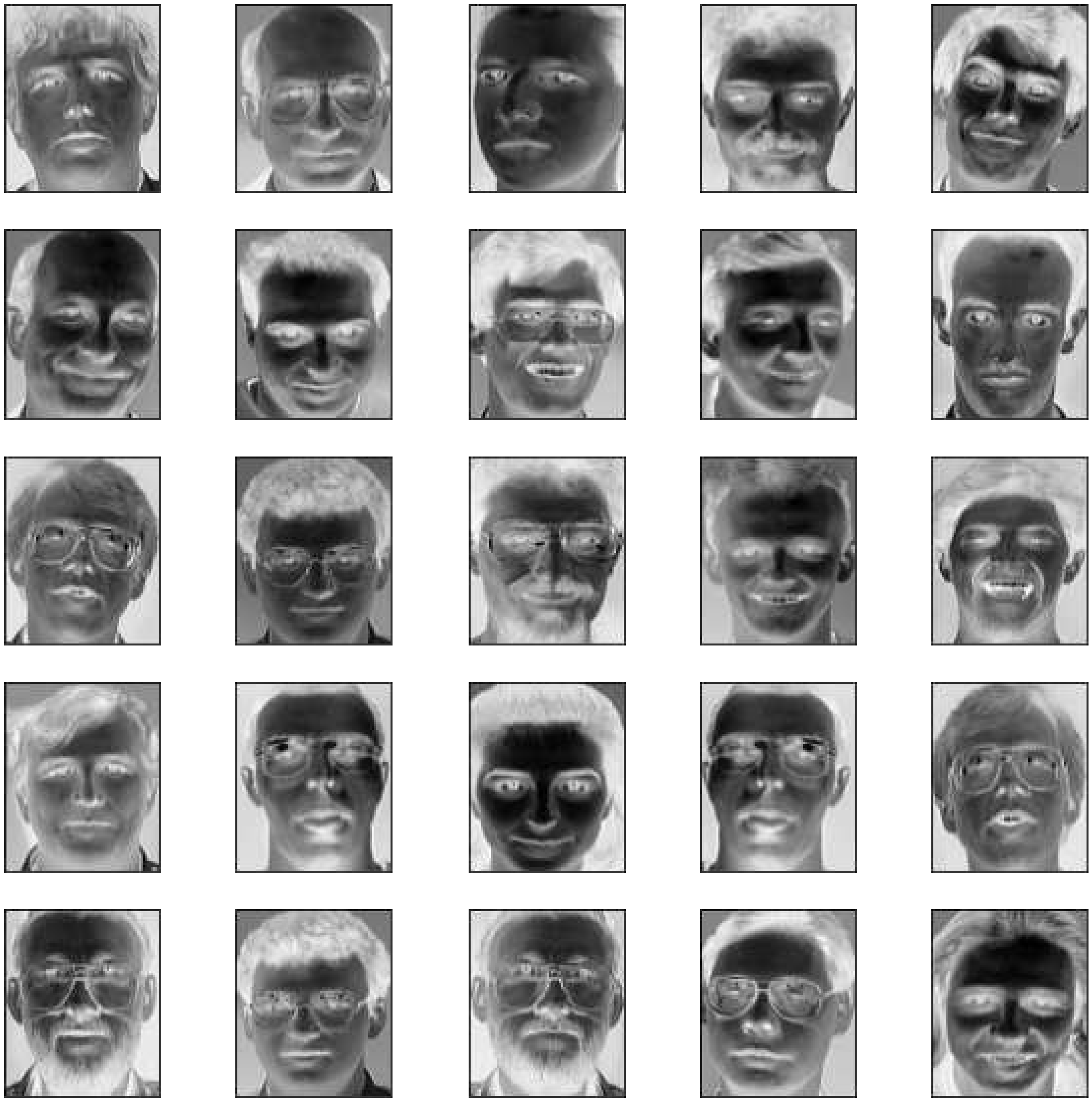
[26]:
```
print('Training dataset modified shape: ',x_train.shape)
print('Testing dataset modified shape: ',x_test.shape)
print('Validating dataset modified shape: ',x_valid.shape)
```

```
Training dataset modified shape:  (216, 112, 92, 1)
Testing dataset modified shape:  (160, 112, 92, 1)
Validating dataset modified shape:  (24, 112, 92, 1)
```

Visualize images in different colormap

[27]:
```
#visualize some images 5 x 5 grid images in gray scale
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i], cmap=plt.cm.binary) # for gray scale
plt.show()
```

```
Exception ignored in: <function NpzFile.__del__ at 0x7f29cc170950>
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/site-packages/numpy/lib/npyio.py", line 230, in
__del__
    self.close()
  File "/usr/local/lib/python3.7/site-packages/numpy/lib/npyio.py", line 221, in
close
    if self.zip is not None:
AttributeError: 'NpzFile' object has no attribute 'zip'
```



[28]:
```
#visualize some images 5 x 5 grid images in autumn
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i],cmap=plt.cm.autumn) # for autumn
plt.show()
```

[29]:
```
#visualize some images 5 x 5 grid images by default
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i])
plt.show()
```

Step 5: Build a CNN model that has 3 main layers:

- Convolutional Layer
- Pooling Layer
- Fully Connected Layer

The objective here is to build and train a CNN model which has accuracy above 90%. It depends upon number of iterations (Epochs) performed and what type of activation function is chosen to train the model. Before deciding the type of activation function chosen for our final model, we will train the model for different types of activation functions and then use that defined function for final prediction.

- Activation functions tested: ['sigmoid', 'relu', 'elu', 'leaky-relu', 'selu']
- For 'selu' (Scaled Exponential Linear Unit), we need to use a kernel initializer 'lecun_normal'

and a special form of dropout 'AlphaDropout()'

[30]:
```
# We will initialize our cnn model with activation function, dropout rate,␣
↪optimizer
def cnn_model(activation,
              dropout_rate,
              optimizer):

    model = Sequential() #initialize Sequential model

    #we created if else version for program to 'selu' version or␣
↪otheractivation functions

    if(activation == 'selu'):
        model.add(Conv2D(32, kernel_size=3,
                  activation=activation,
                  input_shape=image_shape,
                  kernel_initializer='lecun_normal')) #32 filter with kernel␣
↪size of 3 with input shape
        model.add(MaxPooling2D(pool_size=2))

        model.add(Conv2D(64, 3, activation=activation,
                         kernel_initializer='lecun_normal')) #64 filter with␣
↪kernel size of 3 x 3
        model.add(MaxPooling2D(pool_size=2)) #Max pool with size of 2

        model.add(Flatten())
        model.add(Dense(2024, activation=activation,
                        kernel_initializer='lecun_normal'))
        model.add(AlphaDropout(0.5))

        model.add(Dense(1024, activation=activation,
                        kernel_initializer='lecun_normal'))
        model.add(AlphaDropout(0.5))

        model.add(Dense(512, activation=activation,
                        kernel_initializer='lecun_normal'))
        model.add(AlphaDropout(0.5))

        model.add(Dense(20, activation='softmax')) #Output layer
    else:
        model.add(Conv2D(32, kernel_size=3,
                  activation=activation,
                  input_shape=image_shape)) #32 filter with kernel size of 3 x␣
↪3 with input shape
        model.add(MaxPooling2D(pool_size=2))
```

```
        model.add(Conv2D(64,3, activation=activation)) #64 filter with kernel␣
↪size of 3 x 3
        model.add(MaxPooling2D(pool_size=2)) #Max pool with size of 2

        model.add(Flatten())

        model.add(Dense(2024, activation=activation))
        model.add(Dropout(0.5))
        model.add(Dense(1024, activation=activation))
        model.add(Dropout(0.5))
        model.add(Dense(512, activation=activation))
        model.add(Dropout(0.5))

        model.add(Dense(20, activation='softmax')) #Output layer

    model.compile(
        loss='sparse_categorical_crossentropy',
        optimizer=optimizer,
        metrics=['accuracy']
    ) #compile model with loss, optimizer chosen and accuracy as metrics

    return model
```

[31]:
```
#For Leaky-Rely function we need to define aplha parameters using␣
↪get_custom_objects

get_custom_objects().update({'leaky-relu': Activation(LeakyReLU(alpha=0.2))})

# Defining the type of activation functions to be tested
activation_function = ['relu', 'elu', 'leaky-relu', 'selu']
```

```
/usr/local/lib/python3.7/site-packages/keras/activations.py:235: UserWarning: Do
not pass a layer instance (such as LeakyReLU) as the activation argument of
another layer. Instead, advanced activation layers should be used just like any
other layer in a model.
  identifier=identifier.__class__.__name__))
```

Building model and train for all chosen activation functions

[32]:
```
activation_results = [] #creating an empty matrix for storing results for␣
↪activations

for activation in activation_function:
    print('\nTraining with {0} activation function\n'.format(activation))

    model = cnn_model(activation=activation,
                      dropout_rate=0.2,
```

```
                      optimizer=Adam(clipvalue=0.5)) #using 'adam' optimizer␣
↪with clipvalue of 0.5

    history = model.fit(np.array(x_train), np.array(y_train),
                        batch_size=512,
                        epochs=75,
                        verbose=2,
                        validation_data=(np.array(x_valid),np.array(y_valid)))

    activation_results.append(history) #store results

    K.clear_session()
    del model

print(activation_results)
```

```
Training with relu activation function

Train on 216 samples, validate on 24 samples
Epoch 1/75
 - 4s - loss: 2.9945 - accuracy: 0.0324 - val_loss: 3.0009 - val_accuracy:
0.0833
Epoch 2/75
 - 4s - loss: 3.3238 - accuracy: 0.0417 - val_loss: 3.0803 - val_accuracy:
0.0417
Epoch 3/75
 - 4s - loss: 3.3828 - accuracy: 0.0417 - val_loss: 3.0919 - val_accuracy:
0.0000e+00
Epoch 4/75
 - 4s - loss: 3.1760 - accuracy: 0.0648 - val_loss: 3.0564 - val_accuracy:
0.0000e+00
Epoch 5/75
 - 4s - loss: 3.1041 - accuracy: 0.0648 - val_loss: 3.0082 - val_accuracy:
0.0000e+00
Epoch 6/75
 - 4s - loss: 3.0839 - accuracy: 0.0694 - val_loss: 2.9879 - val_accuracy:
0.0417
Epoch 7/75
 - 4s - loss: 2.9970 - accuracy: 0.0602 - val_loss: 2.9854 - val_accuracy:
0.0417
Epoch 8/75
 - 4s - loss: 2.9727 - accuracy: 0.0880 - val_loss: 2.9804 - val_accuracy:
0.0417
Epoch 9/75
 - 4s - loss: 2.9476 - accuracy: 0.1065 - val_loss: 2.9826 - val_accuracy:
0.0000e+00
```

```
Epoch 10/75
 - 4s - loss: 2.9560 - accuracy: 0.0833 - val_loss: 2.9833 - val_accuracy:
0.0000e+00
Epoch 11/75
 - 4s - loss: 2.9219 - accuracy: 0.1250 - val_loss: 2.9814 - val_accuracy:
0.0000e+00
Epoch 12/75
 - 4s - loss: 2.9185 - accuracy: 0.1250 - val_loss: 2.9718 - val_accuracy:
0.0000e+00
Epoch 13/75
 - 4s - loss: 2.9201 - accuracy: 0.1019 - val_loss: 2.9548 - val_accuracy:
0.0000e+00
Epoch 14/75
 - 4s - loss: 2.8430 - accuracy: 0.1435 - val_loss: 2.9390 - val_accuracy:
0.0000e+00
Epoch 15/75
 - 4s - loss: 2.8590 - accuracy: 0.1435 - val_loss: 2.9036 - val_accuracy:
0.0000e+00
Epoch 16/75
 - 4s - loss: 2.7905 - accuracy: 0.1713 - val_loss: 2.8557 - val_accuracy:
0.0417
Epoch 17/75
 - 4s - loss: 2.7226 - accuracy: 0.1713 - val_loss: 2.7934 - val_accuracy:
0.0417
Epoch 18/75
 - 4s - loss: 2.6664 - accuracy: 0.1667 - val_loss: 2.6943 - val_accuracy:
0.0833
Epoch 19/75
 - 4s - loss: 2.6221 - accuracy: 0.1852 - val_loss: 2.5790 - val_accuracy:
0.2083
Epoch 20/75
 - 4s - loss: 2.4742 - accuracy: 0.2639 - val_loss: 2.4230 - val_accuracy:
0.4167
Epoch 21/75
 - 4s - loss: 2.3363 - accuracy: 0.3148 - val_loss: 2.2048 - val_accuracy:
0.5417
Epoch 22/75
 - 4s - loss: 2.1713 - accuracy: 0.3565 - val_loss: 1.8928 - val_accuracy:
0.7083
Epoch 23/75
 - 4s - loss: 1.9891 - accuracy: 0.3380 - val_loss: 1.6414 - val_accuracy:
0.7917
Epoch 24/75
 - 4s - loss: 1.8476 - accuracy: 0.3935 - val_loss: 1.4582 - val_accuracy:
0.8750
Epoch 25/75
 - 4s - loss: 1.5954 - accuracy: 0.5694 - val_loss: 1.1667 - val_accuracy:
0.8750
```

```
Epoch 26/75
 - 4s - loss: 1.4162 - accuracy: 0.5880 - val_loss: 0.9121 - val_accuracy:
0.9167
Epoch 27/75
 - 4s - loss: 1.2480 - accuracy: 0.6204 - val_loss: 0.7463 - val_accuracy:
0.9167
Epoch 28/75
 - 4s - loss: 1.0020 - accuracy: 0.7176 - val_loss: 0.5686 - val_accuracy:
0.9583
Epoch 29/75
 - 4s - loss: 0.8632 - accuracy: 0.7407 - val_loss: 0.4795 - val_accuracy:
0.9167
Epoch 30/75
 - 4s - loss: 0.8397 - accuracy: 0.7083 - val_loss: 0.4654 - val_accuracy:
0.9167
Epoch 31/75
 - 4s - loss: 0.7236 - accuracy: 0.7731 - val_loss: 0.3737 - val_accuracy:
0.9583
Epoch 32/75
 - 4s - loss: 0.5126 - accuracy: 0.8287 - val_loss: 0.3548 - val_accuracy:
0.9167
Epoch 33/75
 - 4s - loss: 0.5642 - accuracy: 0.8333 - val_loss: 0.2660 - val_accuracy:
0.9583
Epoch 34/75
 - 4s - loss: 0.3788 - accuracy: 0.8981 - val_loss: 0.2446 - val_accuracy:
0.9167
Epoch 35/75
 - 4s - loss: 0.3729 - accuracy: 0.8796 - val_loss: 0.1883 - val_accuracy:
0.9583
Epoch 36/75
 - 4s - loss: 0.3589 - accuracy: 0.8981 - val_loss: 0.1651 - val_accuracy:
1.0000
Epoch 37/75
 - 4s - loss: 0.2012 - accuracy: 0.9537 - val_loss: 0.1755 - val_accuracy:
0.9583
Epoch 38/75
 - 4s - loss: 0.1988 - accuracy: 0.9398 - val_loss: 0.1688 - val_accuracy:
0.9167
Epoch 39/75
 - 4s - loss: 0.1635 - accuracy: 0.9352 - val_loss: 0.1271 - val_accuracy:
1.0000
Epoch 40/75
 - 4s - loss: 0.1668 - accuracy: 0.9306 - val_loss: 0.0989 - val_accuracy:
1.0000
Epoch 41/75
 - 4s - loss: 0.1904 - accuracy: 0.9213 - val_loss: 0.0822 - val_accuracy:
1.0000
```

```
Epoch 42/75
 - 4s - loss: 0.1201 - accuracy: 0.9676 - val_loss: 0.0706 - val_accuracy:
1.0000
Epoch 43/75
 - 4s - loss: 0.1228 - accuracy: 0.9537 - val_loss: 0.0608 - val_accuracy:
1.0000
Epoch 44/75
 - 4s - loss: 0.0833 - accuracy: 0.9769 - val_loss: 0.0441 - val_accuracy:
1.0000
Epoch 45/75
 - 4s - loss: 0.0514 - accuracy: 0.9769 - val_loss: 0.0294 - val_accuracy:
1.0000
Epoch 46/75
 - 4s - loss: 0.0864 - accuracy: 0.9722 - val_loss: 0.0215 - val_accuracy:
1.0000
Epoch 47/75
 - 4s - loss: 0.0608 - accuracy: 0.9769 - val_loss: 0.0225 - val_accuracy:
1.0000
Epoch 48/75
 - 4s - loss: 0.0388 - accuracy: 0.9954 - val_loss: 0.0240 - val_accuracy:
1.0000
Epoch 49/75
 - 4s - loss: 0.0463 - accuracy: 0.9861 - val_loss: 0.0213 - val_accuracy:
1.0000
Epoch 50/75
 - 4s - loss: 0.0399 - accuracy: 0.9907 - val_loss: 0.0155 - val_accuracy:
1.0000
Epoch 51/75
 - 4s - loss: 0.0509 - accuracy: 0.9722 - val_loss: 0.0109 - val_accuracy:
1.0000
Epoch 52/75
 - 4s - loss: 0.0323 - accuracy: 0.9907 - val_loss: 0.0099 - val_accuracy:
1.0000
Epoch 53/75
 - 4s - loss: 0.0280 - accuracy: 0.9954 - val_loss: 0.0074 - val_accuracy:
1.0000
Epoch 54/75
 - 4s - loss: 0.0258 - accuracy: 0.9954 - val_loss: 0.0082 - val_accuracy:
1.0000
Epoch 55/75
 - 4s - loss: 0.0104 - accuracy: 1.0000 - val_loss: 0.0136 - val_accuracy:
1.0000
Epoch 56/75
 - 4s - loss: 0.0350 - accuracy: 0.9861 - val_loss: 0.0140 - val_accuracy:
1.0000
Epoch 57/75
 - 4s - loss: 0.0275 - accuracy: 0.9907 - val_loss: 0.0110 - val_accuracy:
1.0000
```

```
Epoch 58/75
 - 4s - loss: 0.0134 - accuracy: 0.9954 - val_loss: 0.0101 - val_accuracy:
1.0000
Epoch 59/75
 - 4s - loss: 0.0224 - accuracy: 0.9954 - val_loss: 0.0103 - val_accuracy:
1.0000
Epoch 60/75
 - 4s - loss: 0.0149 - accuracy: 0.9954 - val_loss: 0.0114 - val_accuracy:
1.0000
Epoch 61/75
 - 4s - loss: 0.0209 - accuracy: 0.9954 - val_loss: 0.0131 - val_accuracy:
1.0000
Epoch 62/75
 - 4s - loss: 0.0179 - accuracy: 0.9954 - val_loss: 0.0199 - val_accuracy:
1.0000
Epoch 63/75
 - 4s - loss: 0.0058 - accuracy: 1.0000 - val_loss: 0.0305 - val_accuracy:
1.0000
Epoch 64/75
 - 4s - loss: 0.0254 - accuracy: 0.9907 - val_loss: 0.0333 - val_accuracy:
1.0000
Epoch 65/75
 - 4s - loss: 0.0090 - accuracy: 1.0000 - val_loss: 0.0322 - val_accuracy:
1.0000
Epoch 66/75
 - 4s - loss: 0.0126 - accuracy: 1.0000 - val_loss: 0.0289 - val_accuracy:
1.0000
Epoch 67/75
 - 4s - loss: 0.0095 - accuracy: 0.9954 - val_loss: 0.0211 - val_accuracy:
1.0000
Epoch 68/75
 - 4s - loss: 0.0180 - accuracy: 0.9954 - val_loss: 0.0139 - val_accuracy:
1.0000
Epoch 69/75
 - 4s - loss: 0.0092 - accuracy: 1.0000 - val_loss: 0.0082 - val_accuracy:
1.0000
Epoch 70/75
 - 4s - loss: 0.0071 - accuracy: 1.0000 - val_loss: 0.0055 - val_accuracy:
1.0000
Epoch 71/75
 - 4s - loss: 0.0117 - accuracy: 1.0000 - val_loss: 0.0039 - val_accuracy:
1.0000
Epoch 72/75
 - 4s - loss: 0.0191 - accuracy: 0.9907 - val_loss: 0.0029 - val_accuracy:
1.0000
Epoch 73/75
 - 4s - loss: 0.0029 - accuracy: 1.0000 - val_loss: 0.0026 - val_accuracy:
1.0000
```

```
Epoch 74/75
 - 4s - loss: 0.0368 - accuracy: 0.9861 - val_loss: 0.0033 - val_accuracy:
1.0000
Epoch 75/75
 - 4s - loss: 0.0124 - accuracy: 0.9954 - val_loss: 0.0051 - val_accuracy:
1.0000

Training with elu activation function

Train on 216 samples, validate on 24 samples
Epoch 1/75
 - 5s - loss: 3.1566 - accuracy: 0.0324 - val_loss: 5.6138 - val_accuracy:
0.0417
Epoch 2/75
 - 5s - loss: 9.1273 - accuracy: 0.0463 - val_loss: 3.9957 - val_accuracy:
0.0833
Epoch 3/75
 - 5s - loss: 8.6221 - accuracy: 0.0694 - val_loss: 3.5591 - val_accuracy:
0.1250
Epoch 4/75
 - 5s - loss: 5.0292 - accuracy: 0.0694 - val_loss: 9.0853 - val_accuracy:
0.0000e+00
Epoch 5/75
 - 5s - loss: 8.6928 - accuracy: 0.0741 - val_loss: 5.8023 - val_accuracy:
0.0000e+00
Epoch 6/75
 - 5s - loss: 9.0991 - accuracy: 0.0787 - val_loss: 4.3718 - val_accuracy:
0.0000e+00
Epoch 7/75
 - 5s - loss: 5.5316 - accuracy: 0.1111 - val_loss: 6.8534 - val_accuracy:
0.0000e+00
Epoch 8/75
 - 5s - loss: 6.7668 - accuracy: 0.0694 - val_loss: 6.3118 - val_accuracy:
0.0000e+00
Epoch 9/75
 - 5s - loss: 6.2607 - accuracy: 0.0880 - val_loss: 2.8033 - val_accuracy:
0.1250
Epoch 10/75
 - 5s - loss: 4.8520 - accuracy: 0.1111 - val_loss: 2.8673 - val_accuracy:
0.2083
Epoch 11/75
 - 5s - loss: 4.8594 - accuracy: 0.1019 - val_loss: 2.4578 - val_accuracy:
0.1250
Epoch 12/75
 - 5s - loss: 3.8156 - accuracy: 0.1574 - val_loss: 2.7561 - val_accuracy:
0.1667
Epoch 13/75
 - 5s - loss: 3.5186 - accuracy: 0.1713 - val_loss: 3.5073 - val_accuracy:
```

```
0.0417
Epoch 14/75
 - 5s - loss: 3.0790 - accuracy: 0.2546 - val_loss: 1.9228 - val_accuracy:
0.2500
Epoch 15/75
 - 5s - loss: 2.6147 - accuracy: 0.3102 - val_loss: 1.4267 - val_accuracy:
0.6250
Epoch 16/75
 - 5s - loss: 1.8780 - accuracy: 0.4491 - val_loss: 1.5616 - val_accuracy:
0.4583
Epoch 17/75
 - 5s - loss: 2.4683 - accuracy: 0.4120 - val_loss: 1.9295 - val_accuracy:
0.3750
Epoch 18/75
 - 5s - loss: 2.1440 - accuracy: 0.4398 - val_loss: 0.8852 - val_accuracy:
0.7917
Epoch 19/75
 - 5s - loss: 1.2032 - accuracy: 0.6157 - val_loss: 0.8614 - val_accuracy:
0.6667
Epoch 20/75
 - 5s - loss: 1.2626 - accuracy: 0.5741 - val_loss: 1.2117 - val_accuracy:
0.5417
Epoch 21/75
 - 5s - loss: 1.4170 - accuracy: 0.5509 - val_loss: 0.7973 - val_accuracy:
0.7500
Epoch 22/75
 - 5s - loss: 0.8997 - accuracy: 0.7269 - val_loss: 0.7483 - val_accuracy:
0.7083
Epoch 23/75
 - 5s - loss: 1.0084 - accuracy: 0.6898 - val_loss: 0.5409 - val_accuracy:
0.9167
Epoch 24/75
 - 5s - loss: 0.8658 - accuracy: 0.7546 - val_loss: 0.4683 - val_accuracy:
0.9167
Epoch 25/75
 - 5s - loss: 0.5193 - accuracy: 0.8565 - val_loss: 1.2764 - val_accuracy:
0.7083
Epoch 26/75
 - 5s - loss: 0.7733 - accuracy: 0.7639 - val_loss: 0.5556 - val_accuracy:
0.8333
Epoch 27/75
 - 5s - loss: 0.3232 - accuracy: 0.9120 - val_loss: 0.4555 - val_accuracy:
0.9167
Epoch 28/75
 - 5s - loss: 0.3741 - accuracy: 0.8889 - val_loss: 0.3932 - val_accuracy:
0.8750
Epoch 29/75
 - 5s - loss: 0.2397 - accuracy: 0.9259 - val_loss: 0.4508 - val_accuracy:
```

```
0.8750
Epoch 30/75
 - 5s - loss: 0.3307 - accuracy: 0.9120 - val_loss: 0.1996 - val_accuracy:
0.9167
Epoch 31/75
 - 5s - loss: 0.1370 - accuracy: 0.9583 - val_loss: 0.2916 - val_accuracy:
0.9167
Epoch 32/75
 - 5s - loss: 0.2671 - accuracy: 0.9074 - val_loss: 0.1434 - val_accuracy:
0.9583
Epoch 33/75
 - 5s - loss: 0.1422 - accuracy: 0.9583 - val_loss: 0.2452 - val_accuracy:
0.9167
Epoch 34/75
 - 5s - loss: 0.1561 - accuracy: 0.9537 - val_loss: 0.1550 - val_accuracy:
0.9583
Epoch 35/75
 - 5s - loss: 0.1630 - accuracy: 0.9398 - val_loss: 0.1683 - val_accuracy:
0.9583
Epoch 36/75
 - 5s - loss: 0.1042 - accuracy: 0.9722 - val_loss: 0.1642 - val_accuracy:
0.9583
Epoch 37/75
 - 5s - loss: 0.1056 - accuracy: 0.9815 - val_loss: 0.0955 - val_accuracy:
1.0000
Epoch 38/75
 - 5s - loss: 0.1091 - accuracy: 0.9583 - val_loss: 0.1263 - val_accuracy:
0.9583
Epoch 39/75
 - 5s - loss: 0.0584 - accuracy: 0.9861 - val_loss: 0.1457 - val_accuracy:
0.9583
Epoch 40/75
 - 5s - loss: 0.0599 - accuracy: 0.9815 - val_loss: 0.1025 - val_accuracy:
0.9583
Epoch 41/75
 - 5s - loss: 0.0481 - accuracy: 0.9815 - val_loss: 0.0492 - val_accuracy:
1.0000
Epoch 42/75
 - 5s - loss: 0.0272 - accuracy: 0.9907 - val_loss: 0.0577 - val_accuracy:
1.0000
Epoch 43/75
 - 5s - loss: 0.0359 - accuracy: 0.9954 - val_loss: 0.0642 - val_accuracy:
1.0000
Epoch 44/75
 - 5s - loss: 0.0280 - accuracy: 0.9954 - val_loss: 0.0416 - val_accuracy:
1.0000
Epoch 45/75
 - 5s - loss: 0.0154 - accuracy: 1.0000 - val_loss: 0.0289 - val_accuracy:
```

```
1.0000
Epoch 46/75
 - 5s - loss: 0.0215 - accuracy: 0.9954 - val_loss: 0.0284 - val_accuracy:
1.0000
Epoch 47/75
 - 5s - loss: 0.0195 - accuracy: 1.0000 - val_loss: 0.0345 - val_accuracy:
1.0000
Epoch 48/75
 - 5s - loss: 0.0201 - accuracy: 0.9907 - val_loss: 0.0445 - val_accuracy:
1.0000
Epoch 49/75
 - 5s - loss: 0.0178 - accuracy: 0.9954 - val_loss: 0.0607 - val_accuracy:
0.9583
Epoch 50/75
 - 5s - loss: 0.0071 - accuracy: 1.0000 - val_loss: 0.0746 - val_accuracy:
0.9583
Epoch 51/75
 - 5s - loss: 0.0161 - accuracy: 1.0000 - val_loss: 0.0675 - val_accuracy:
0.9583
Epoch 52/75
 - 5s - loss: 0.0139 - accuracy: 1.0000 - val_loss: 0.0453 - val_accuracy:
1.0000
Epoch 53/75
 - 5s - loss: 0.0230 - accuracy: 0.9861 - val_loss: 0.0208 - val_accuracy:
1.0000
Epoch 54/75
 - 5s - loss: 0.0114 - accuracy: 1.0000 - val_loss: 0.0100 - val_accuracy:
1.0000
Epoch 55/75
 - 5s - loss: 0.0110 - accuracy: 1.0000 - val_loss: 0.0060 - val_accuracy:
1.0000
Epoch 56/75
 - 5s - loss: 0.0110 - accuracy: 1.0000 - val_loss: 0.0054 - val_accuracy:
1.0000
Epoch 57/75
 - 5s - loss: 0.0398 - accuracy: 0.9861 - val_loss: 0.0082 - val_accuracy:
1.0000
Epoch 58/75
 - 5s - loss: 0.0085 - accuracy: 0.9954 - val_loss: 0.0159 - val_accuracy:
1.0000
Epoch 59/75
 - 5s - loss: 0.0069 - accuracy: 1.0000 - val_loss: 0.0284 - val_accuracy:
1.0000
Epoch 60/75
 - 5s - loss: 0.0153 - accuracy: 1.0000 - val_loss: 0.0229 - val_accuracy:
1.0000
Epoch 61/75
 - 5s - loss: 0.0106 - accuracy: 1.0000 - val_loss: 0.0161 - val_accuracy:
```

```
1.0000
Epoch 62/75
 - 5s - loss: 0.0107 - accuracy: 1.0000 - val_loss: 0.0120 - val_accuracy:
1.0000
Epoch 63/75
 - 5s - loss: 0.0214 - accuracy: 0.9954 - val_loss: 0.0078 - val_accuracy:
1.0000
Epoch 64/75
 - 5s - loss: 0.0162 - accuracy: 0.9954 - val_loss: 0.0081 - val_accuracy:
1.0000
Epoch 65/75
 - 5s - loss: 0.0052 - accuracy: 1.0000 - val_loss: 0.0108 - val_accuracy:
1.0000
Epoch 66/75
 - 5s - loss: 0.0031 - accuracy: 1.0000 - val_loss: 0.0158 - val_accuracy:
1.0000
Epoch 67/75
 - 5s - loss: 0.0039 - accuracy: 1.0000 - val_loss: 0.0238 - val_accuracy:
1.0000
Epoch 68/75
 - 5s - loss: 0.0141 - accuracy: 0.9954 - val_loss: 0.0204 - val_accuracy:
1.0000
Epoch 69/75
 - 5s - loss: 0.0048 - accuracy: 1.0000 - val_loss: 0.0120 - val_accuracy:
1.0000
Epoch 70/75
 - 5s - loss: 0.0063 - accuracy: 1.0000 - val_loss: 0.0054 - val_accuracy:
1.0000
Epoch 71/75
 - 5s - loss: 0.0047 - accuracy: 1.0000 - val_loss: 0.0036 - val_accuracy:
1.0000
Epoch 72/75
 - 5s - loss: 0.0060 - accuracy: 1.0000 - val_loss: 0.0035 - val_accuracy:
1.0000
Epoch 73/75
 - 5s - loss: 0.0047 - accuracy: 1.0000 - val_loss: 0.0034 - val_accuracy:
1.0000
Epoch 74/75
 - 5s - loss: 0.0036 - accuracy: 1.0000 - val_loss: 0.0036 - val_accuracy:
1.0000
Epoch 75/75
 - 5s - loss: 0.0055 - accuracy: 0.9954 - val_loss: 0.0037 - val_accuracy:
1.0000

Training with leaky-relu activation function

Train on 216 samples, validate on 24 samples
Epoch 1/75
```

```
 - 5s - loss: 3.0096 - accuracy: 0.0463 - val_loss: 3.3727 - val_accuracy:
0.0000e+00
Epoch 2/75
 - 4s - loss: 3.8225 - accuracy: 0.0278 - val_loss: 2.9893 - val_accuracy:
0.0833
Epoch 3/75
 - 4s - loss: 3.3211 - accuracy: 0.0694 - val_loss: 2.9648 - val_accuracy:
0.1667
Epoch 4/75
 - 4s - loss: 2.9551 - accuracy: 0.0880 - val_loss: 2.9355 - val_accuracy:
0.0833
Epoch 5/75
 - 4s - loss: 2.8489 - accuracy: 0.1898 - val_loss: 2.9719 - val_accuracy:
0.0833
Epoch 6/75
 - 4s - loss: 3.1739 - accuracy: 0.0926 - val_loss: 2.7812 - val_accuracy:
0.1667
Epoch 7/75
 - 4s - loss: 2.7697 - accuracy: 0.1667 - val_loss: 2.7949 - val_accuracy:
0.1250
Epoch 8/75
 - 4s - loss: 2.6586 - accuracy: 0.2222 - val_loss: 2.7655 - val_accuracy:
0.0833
Epoch 9/75
 - 4s - loss: 2.6006 - accuracy: 0.2731 - val_loss: 2.6247 - val_accuracy:
0.5833
Epoch 10/75
 - 4s - loss: 2.3982 - accuracy: 0.3750 - val_loss: 2.4207 - val_accuracy:
0.7500
Epoch 11/75
 - 4s - loss: 2.2288 - accuracy: 0.4444 - val_loss: 2.1437 - val_accuracy:
0.7083
Epoch 12/75
 - 4s - loss: 2.0684 - accuracy: 0.4676 - val_loss: 1.8879 - val_accuracy:
0.7083
Epoch 13/75
 - 4s - loss: 1.7415 - accuracy: 0.5694 - val_loss: 1.6051 - val_accuracy:
0.7500
Epoch 14/75
 - 4s - loss: 1.4778 - accuracy: 0.6528 - val_loss: 1.3095 - val_accuracy:
0.7500
Epoch 15/75
 - 4s - loss: 1.1482 - accuracy: 0.7407 - val_loss: 0.9112 - val_accuracy:
0.7917
Epoch 16/75
 - 4s - loss: 0.8516 - accuracy: 0.7639 - val_loss: 0.6149 - val_accuracy:
1.0000
Epoch 17/75
```

```
 - 4s - loss: 0.6604 - accuracy: 0.8333 - val_loss: 0.5101 - val_accuracy:
0.9583
Epoch 18/75
 - 4s - loss: 0.5878 - accuracy: 0.8333 - val_loss: 0.4523 - val_accuracy:
0.8333
Epoch 19/75
 - 4s - loss: 0.4290 - accuracy: 0.8704 - val_loss: 0.3279 - val_accuracy:
0.9583
Epoch 20/75
 - 4s - loss: 0.2896 - accuracy: 0.9213 - val_loss: 0.2276 - val_accuracy:
1.0000
Epoch 21/75
 - 4s - loss: 0.2503 - accuracy: 0.9259 - val_loss: 0.2022 - val_accuracy:
0.9583
Epoch 22/75
 - 4s - loss: 0.2512 - accuracy: 0.9491 - val_loss: 0.1949 - val_accuracy:
1.0000
Epoch 23/75
 - 4s - loss: 0.1475 - accuracy: 0.9722 - val_loss: 0.1616 - val_accuracy:
0.9583
Epoch 24/75
 - 4s - loss: 0.1261 - accuracy: 0.9769 - val_loss: 0.1390 - val_accuracy:
0.9583
Epoch 25/75
 - 4s - loss: 0.1346 - accuracy: 0.9583 - val_loss: 0.2069 - val_accuracy:
0.9167
Epoch 26/75
 - 4s - loss: 0.1348 - accuracy: 0.9583 - val_loss: 0.0835 - val_accuracy:
1.0000
Epoch 27/75
 - 4s - loss: 0.0524 - accuracy: 0.9815 - val_loss: 0.0456 - val_accuracy:
1.0000
Epoch 28/75
 - 4s - loss: 0.0504 - accuracy: 0.9815 - val_loss: 0.0484 - val_accuracy:
1.0000
Epoch 29/75
 - 4s - loss: 0.0596 - accuracy: 0.9815 - val_loss: 0.0453 - val_accuracy:
1.0000
Epoch 30/75
 - 4s - loss: 0.0422 - accuracy: 0.9861 - val_loss: 0.0947 - val_accuracy:
0.9583
Epoch 31/75
 - 4s - loss: 0.0278 - accuracy: 0.9907 - val_loss: 0.1364 - val_accuracy:
0.9167
Epoch 32/75
 - 4s - loss: 0.0270 - accuracy: 0.9954 - val_loss: 0.1336 - val_accuracy:
0.9167
Epoch 33/75
```

```
 - 4s - loss: 0.0168 - accuracy: 1.0000 - val_loss: 0.1216 - val_accuracy:
0.9167
Epoch 34/75
 - 4s - loss: 0.0404 - accuracy: 0.9815 - val_loss: 0.1024 - val_accuracy:
0.9583
Epoch 35/75
 - 4s - loss: 0.0078 - accuracy: 1.0000 - val_loss: 0.1204 - val_accuracy:
0.9583
Epoch 36/75
 - 4s - loss: 0.0429 - accuracy: 0.9861 - val_loss: 0.0735 - val_accuracy:
0.9583
Epoch 37/75
 - 4s - loss: 0.0466 - accuracy: 0.9815 - val_loss: 0.0216 - val_accuracy:
1.0000
Epoch 38/75
 - 4s - loss: 0.0061 - accuracy: 1.0000 - val_loss: 0.1037 - val_accuracy:
0.9583
Epoch 39/75
 - 4s - loss: 0.0152 - accuracy: 0.9954 - val_loss: 0.1710 - val_accuracy:
0.9583
Epoch 40/75
 - 4s - loss: 0.0431 - accuracy: 0.9907 - val_loss: 0.0578 - val_accuracy:
0.9583
Epoch 41/75
 - 4s - loss: 0.0137 - accuracy: 1.0000 - val_loss: 0.0112 - val_accuracy:
1.0000
Epoch 42/75
 - 4s - loss: 0.0060 - accuracy: 1.0000 - val_loss: 0.0103 - val_accuracy:
1.0000
Epoch 43/75
 - 4s - loss: 0.0083 - accuracy: 1.0000 - val_loss: 0.0202 - val_accuracy:
1.0000
Epoch 44/75
 - 4s - loss: 0.0161 - accuracy: 0.9907 - val_loss: 0.0176 - val_accuracy:
1.0000
Epoch 45/75
 - 4s - loss: 0.0157 - accuracy: 0.9907 - val_loss: 0.0056 - val_accuracy:
1.0000
Epoch 46/75
 - 4s - loss: 0.0040 - accuracy: 1.0000 - val_loss: 0.0026 - val_accuracy:
1.0000
Epoch 47/75
 - 4s - loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.0022 - val_accuracy:
1.0000
Epoch 48/75
 - 4s - loss: 0.0060 - accuracy: 0.9954 - val_loss: 0.0027 - val_accuracy:
1.0000
Epoch 49/75
```

```
 - 4s - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.0050 - val_accuracy:
1.0000
Epoch 50/75
 - 4s - loss: 0.0038 - accuracy: 1.0000 - val_loss: 0.0082 - val_accuracy:
1.0000
Epoch 51/75
 - 4s - loss: 0.0037 - accuracy: 1.0000 - val_loss: 0.0100 - val_accuracy:
1.0000
Epoch 52/75
 - 4s - loss: 0.0061 - accuracy: 1.0000 - val_loss: 0.0086 - val_accuracy:
1.0000
Epoch 53/75
 - 4s - loss: 0.0018 - accuracy: 1.0000 - val_loss: 0.0085 - val_accuracy:
1.0000
Epoch 54/75
 - 4s - loss: 0.0048 - accuracy: 1.0000 - val_loss: 0.0089 - val_accuracy:
1.0000
Epoch 55/75
 - 4s - loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.0112 - val_accuracy:
1.0000
Epoch 56/75
 - 4s - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.0155 - val_accuracy:
1.0000
Epoch 57/75
 - 4s - loss: 0.0030 - accuracy: 1.0000 - val_loss: 0.0195 - val_accuracy:
1.0000
Epoch 58/75
 - 4s - loss: 0.0050 - accuracy: 1.0000 - val_loss: 0.0157 - val_accuracy:
1.0000
Epoch 59/75
 - 4s - loss: 0.0038 - accuracy: 1.0000 - val_loss: 0.0102 - val_accuracy:
1.0000
Epoch 60/75
 - 4s - loss: 0.0052 - accuracy: 1.0000 - val_loss: 0.0068 - val_accuracy:
1.0000
Epoch 61/75
 - 5s - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.0048 - val_accuracy:
1.0000
Epoch 62/75
 - 4s - loss: 0.0012 - accuracy: 1.0000 - val_loss: 0.0038 - val_accuracy:
1.0000
Epoch 63/75
 - 4s - loss: 6.5235e-04 - accuracy: 1.0000 - val_loss: 0.0032 - val_accuracy:
1.0000
Epoch 64/75
 - 4s - loss: 4.7098e-04 - accuracy: 1.0000 - val_loss: 0.0028 - val_accuracy:
1.0000
Epoch 65/75
```

```
 - 4s - loss: 5.9693e-04 - accuracy: 1.0000 - val_loss: 0.0025 - val_accuracy:
1.0000
Epoch 66/75
 - 4s - loss: 0.0017 - accuracy: 1.0000 - val_loss: 0.0022 - val_accuracy:
1.0000
Epoch 67/75
 - 4s - loss: 0.0071 - accuracy: 0.9954 - val_loss: 0.0019 - val_accuracy:
1.0000
Epoch 68/75
 - 4s - loss: 0.0040 - accuracy: 1.0000 - val_loss: 0.0021 - val_accuracy:
1.0000
Epoch 69/75
 - 4s - loss: 2.5035e-04 - accuracy: 1.0000 - val_loss: 0.0024 - val_accuracy:
1.0000
Epoch 70/75
 - 4s - loss: 7.0766e-04 - accuracy: 1.0000 - val_loss: 0.0027 - val_accuracy:
1.0000
Epoch 71/75
 - 4s - loss: 4.9379e-04 - accuracy: 1.0000 - val_loss: 0.0031 - val_accuracy:
1.0000
Epoch 72/75
 - 4s - loss: 7.4423e-04 - accuracy: 1.0000 - val_loss: 0.0034 - val_accuracy:
1.0000
Epoch 73/75
 - 4s - loss: 3.9189e-04 - accuracy: 1.0000 - val_loss: 0.0039 - val_accuracy:
1.0000
Epoch 74/75
 - 4s - loss: 0.0019 - accuracy: 1.0000 - val_loss: 0.0048 - val_accuracy:
1.0000
Epoch 75/75
 - 4s - loss: 0.0241 - accuracy: 0.9954 - val_loss: 0.0059 - val_accuracy:
1.0000

Training with selu activation function

Train on 216 samples, validate on 24 samples
Epoch 1/75
 - 7s - loss: 3.8140 - accuracy: 0.0509 - val_loss: 14.4682 - val_accuracy:
0.0833
Epoch 2/75
 - 6s - loss: 5.8868 - accuracy: 0.0509 - val_loss: 13.9102 - val_accuracy:
0.1667
Epoch 3/75
 - 6s - loss: 5.8033 - accuracy: 0.0509 - val_loss: 9.6964 - val_accuracy:
0.0833
Epoch 4/75
 - 6s - loss: 4.7168 - accuracy: 0.0509 - val_loss: 7.9077 - val_accuracy:
0.0417
```

```
Epoch 5/75
 - 6s - loss: 4.1477 - accuracy: 0.0417 - val_loss: 6.7984 - val_accuracy:
0.0417
Epoch 6/75
 - 6s - loss: 3.7114 - accuracy: 0.0741 - val_loss: 39.7962 - val_accuracy:
0.0833
Epoch 7/75
 - 6s - loss: 8.0736 - accuracy: 0.0509 - val_loss: 9.8940 - val_accuracy:
0.0000e+00
Epoch 8/75
 - 6s - loss: 4.0603 - accuracy: 0.0509 - val_loss: 5.4982 - val_accuracy:
0.0417
Epoch 9/75
 - 6s - loss: 3.6581 - accuracy: 0.0648 - val_loss: 4.5443 - val_accuracy:
0.1667
Epoch 10/75
 - 6s - loss: 3.6891 - accuracy: 0.0741 - val_loss: 4.6933 - val_accuracy:
0.0417
Epoch 11/75
 - 6s - loss: 3.6950 - accuracy: 0.0602 - val_loss: 5.0922 - val_accuracy:
0.0417
Epoch 12/75
 - 6s - loss: 3.6696 - accuracy: 0.0880 - val_loss: 4.3342 - val_accuracy:
0.0417
Epoch 13/75
 - 6s - loss: 3.5122 - accuracy: 0.0694 - val_loss: 3.8978 - val_accuracy:
0.0417
Epoch 14/75
 - 6s - loss: 3.4849 - accuracy: 0.1111 - val_loss: 3.4533 - val_accuracy:
0.0000e+00
Epoch 15/75
 - 6s - loss: 3.4759 - accuracy: 0.0787 - val_loss: 3.1833 - val_accuracy:
0.1250
Epoch 16/75
 - 6s - loss: 3.4349 - accuracy: 0.0787 - val_loss: 3.2155 - val_accuracy:
0.1250
Epoch 17/75
 - 6s - loss: 3.3498 - accuracy: 0.0880 - val_loss: 3.6055 - val_accuracy:
0.1250
Epoch 18/75
 - 6s - loss: 3.2026 - accuracy: 0.1250 - val_loss: 3.9758 - val_accuracy:
0.1250
Epoch 19/75
 - 6s - loss: 3.1111 - accuracy: 0.1157 - val_loss: 4.0885 - val_accuracy:
0.1250
Epoch 20/75
 - 6s - loss: 3.0948 - accuracy: 0.1389 - val_loss: 4.1561 - val_accuracy:
0.0000e+00
```

```
Epoch 21/75
 - 6s - loss: 2.8501 - accuracy: 0.1343 - val_loss: 4.1883 - val_accuracy:
0.0000e+00
Epoch 22/75
 - 6s - loss: 2.7838 - accuracy: 0.1389 - val_loss: 3.9687 - val_accuracy:
0.0000e+00
Epoch 23/75
 - 6s - loss: 2.7180 - accuracy: 0.1944 - val_loss: 3.9069 - val_accuracy:
0.0000e+00
Epoch 24/75
 - 6s - loss: 2.6329 - accuracy: 0.1944 - val_loss: 3.8626 - val_accuracy:
0.0000e+00
Epoch 25/75
 - 6s - loss: 2.5359 - accuracy: 0.2500 - val_loss: 4.1352 - val_accuracy:
0.1250
Epoch 26/75
 - 6s - loss: 2.4492 - accuracy: 0.2685 - val_loss: 3.9574 - val_accuracy:
0.1250
Epoch 27/75
 - 6s - loss: 2.2754 - accuracy: 0.3241 - val_loss: 2.8474 - val_accuracy:
0.0833
Epoch 28/75
 - 6s - loss: 2.1595 - accuracy: 0.3056 - val_loss: 2.3147 - val_accuracy:
0.3750
Epoch 29/75
 - 6s - loss: 2.1416 - accuracy: 0.3194 - val_loss: 2.2493 - val_accuracy:
0.3750
Epoch 30/75
 - 6s - loss: 2.0096 - accuracy: 0.3981 - val_loss: 2.4726 - val_accuracy:
0.2500
Epoch 31/75
 - 6s - loss: 1.9616 - accuracy: 0.3843 - val_loss: 2.5722 - val_accuracy:
0.3750
Epoch 32/75
 - 6s - loss: 1.9190 - accuracy: 0.3935 - val_loss: 2.2946 - val_accuracy:
0.3750
Epoch 33/75
 - 6s - loss: 1.7682 - accuracy: 0.4167 - val_loss: 1.6627 - val_accuracy:
0.5000
Epoch 34/75
 - 6s - loss: 1.6244 - accuracy: 0.4722 - val_loss: 1.3250 - val_accuracy:
0.5833
Epoch 35/75
 - 6s - loss: 1.4603 - accuracy: 0.5324 - val_loss: 1.1094 - val_accuracy:
0.6667
Epoch 36/75
 - 6s - loss: 1.3694 - accuracy: 0.5417 - val_loss: 0.8185 - val_accuracy:
0.7917
```

```
Epoch 37/75
 - 6s - loss: 1.2521 - accuracy: 0.5972 - val_loss: 0.7402 - val_accuracy:
0.8750
Epoch 38/75
 - 6s - loss: 1.1174 - accuracy: 0.6528 - val_loss: 0.6583 - val_accuracy:
0.8750
Epoch 39/75
 - 6s - loss: 1.1389 - accuracy: 0.6204 - val_loss: 0.8012 - val_accuracy:
0.9167
Epoch 40/75
 - 6s - loss: 0.9493 - accuracy: 0.7222 - val_loss: 0.3887 - val_accuracy:
0.8750
Epoch 41/75
 - 6s - loss: 1.0138 - accuracy: 0.6620 - val_loss: 0.7291 - val_accuracy:
0.9167
Epoch 42/75
 - 6s - loss: 0.7624 - accuracy: 0.7269 - val_loss: 0.4266 - val_accuracy:
0.9167
Epoch 43/75
 - 6s - loss: 0.7479 - accuracy: 0.7685 - val_loss: 0.5096 - val_accuracy:
0.9167
Epoch 44/75
 - 6s - loss: 0.6092 - accuracy: 0.8102 - val_loss: 0.8282 - val_accuracy:
0.9167
Epoch 45/75
 - 6s - loss: 0.5636 - accuracy: 0.8611 - val_loss: 0.5003 - val_accuracy:
0.9167
Epoch 46/75
 - 6s - loss: 0.4417 - accuracy: 0.8565 - val_loss: 0.4074 - val_accuracy:
0.9167
Epoch 47/75
 - 6s - loss: 0.4663 - accuracy: 0.8380 - val_loss: 0.8148 - val_accuracy:
0.9167
Epoch 48/75
 - 6s - loss: 0.4011 - accuracy: 0.8843 - val_loss: 0.6267 - val_accuracy:
0.9167
Epoch 49/75
 - 6s - loss: 0.3601 - accuracy: 0.8704 - val_loss: 0.2406 - val_accuracy:
0.9167
Epoch 50/75
 - 6s - loss: 0.3372 - accuracy: 0.8843 - val_loss: 0.2052 - val_accuracy:
0.9583
Epoch 51/75
 - 6s - loss: 0.2975 - accuracy: 0.9074 - val_loss: 0.0944 - val_accuracy:
0.9583
Epoch 52/75
 - 6s - loss: 0.3561 - accuracy: 0.8796 - val_loss: 0.0930 - val_accuracy:
0.9583
```

```
Epoch 53/75
 - 6s - loss: 0.2253 - accuracy: 0.9444 - val_loss: 0.2721 - val_accuracy:
0.9167
Epoch 54/75
 - 6s - loss: 0.2320 - accuracy: 0.9491 - val_loss: 0.3894 - val_accuracy:
0.9583
Epoch 55/75
 - 6s - loss: 0.1626 - accuracy: 0.9491 - val_loss: 0.3176 - val_accuracy:
0.9167
Epoch 56/75
 - 6s - loss: 0.1978 - accuracy: 0.9491 - val_loss: 0.0021 - val_accuracy:
1.0000
Epoch 57/75
 - 6s - loss: 0.1235 - accuracy: 0.9583 - val_loss: 7.5935e-04 - val_accuracy:
1.0000
Epoch 58/75
 - 6s - loss: 0.1448 - accuracy: 0.9630 - val_loss: 0.0303 - val_accuracy:
0.9583
Epoch 59/75
 - 6s - loss: 0.0910 - accuracy: 0.9722 - val_loss: 0.0339 - val_accuracy:
0.9583
Epoch 60/75
 - 6s - loss: 0.1068 - accuracy: 0.9583 - val_loss: 1.0828e-06 - val_accuracy:
1.0000
Epoch 61/75
 - 6s - loss: 0.0672 - accuracy: 0.9861 - val_loss: 1.2914e-07 - val_accuracy:
1.0000
Epoch 62/75
 - 6s - loss: 0.1115 - accuracy: 0.9491 - val_loss: 1.1672e-06 - val_accuracy:
1.0000
Epoch 63/75
 - 6s - loss: 0.0873 - accuracy: 0.9861 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
Epoch 64/75
 - 6s - loss: 0.0596 - accuracy: 0.9769 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
Epoch 65/75
 - 6s - loss: 0.0640 - accuracy: 0.9861 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
Epoch 66/75
 - 6s - loss: 0.0467 - accuracy: 0.9907 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
Epoch 67/75
 - 6s - loss: 0.0249 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
Epoch 68/75
 - 6s - loss: 0.0291 - accuracy: 0.9954 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
```

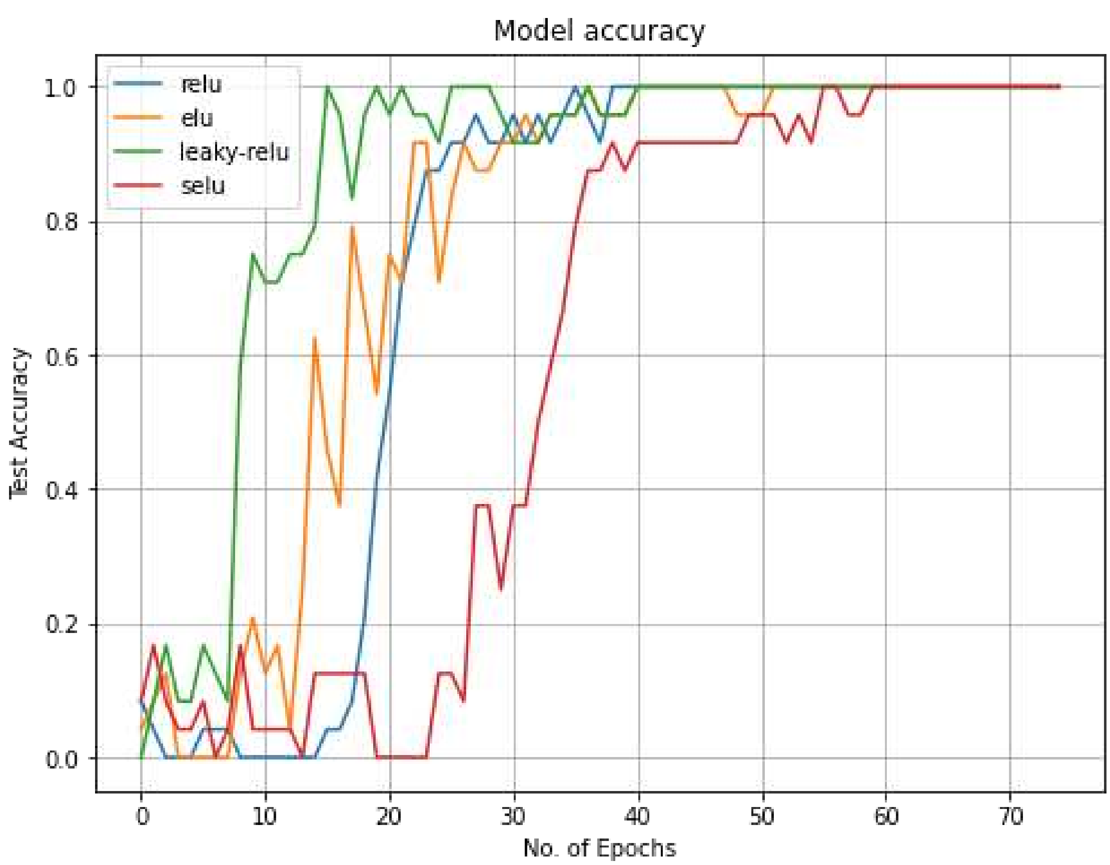
```
Epoch 69/75
 - 6s - loss: 0.0657 - accuracy: 0.9815 - val_loss: 4.9671e-09 - val_accuracy:
1.0000
Epoch 70/75
 - 6s - loss: 0.0547 - accuracy: 0.9815 - val_loss: 4.9671e-09 - val_accuracy:
1.0000
Epoch 71/75
 - 6s - loss: 0.0400 - accuracy: 0.9815 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
Epoch 72/75
 - 6s - loss: 0.0332 - accuracy: 0.9907 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
Epoch 73/75
 - 6s - loss: 0.0369 - accuracy: 0.9954 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
Epoch 74/75
 - 6s - loss: 0.0231 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
Epoch 75/75
 - 6s - loss: 0.0184 - accuracy: 0.9954 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
[<keras.callbacks.callbacks.History object at 0x7f29701eb690>,
<keras.callbacks.callbacks.History object at 0x7f297073de50>,
<keras.callbacks.callbacks.History object at 0x7f2964760890>,
<keras.callbacks.callbacks.History object at 0x7f297b02ff10>]
```

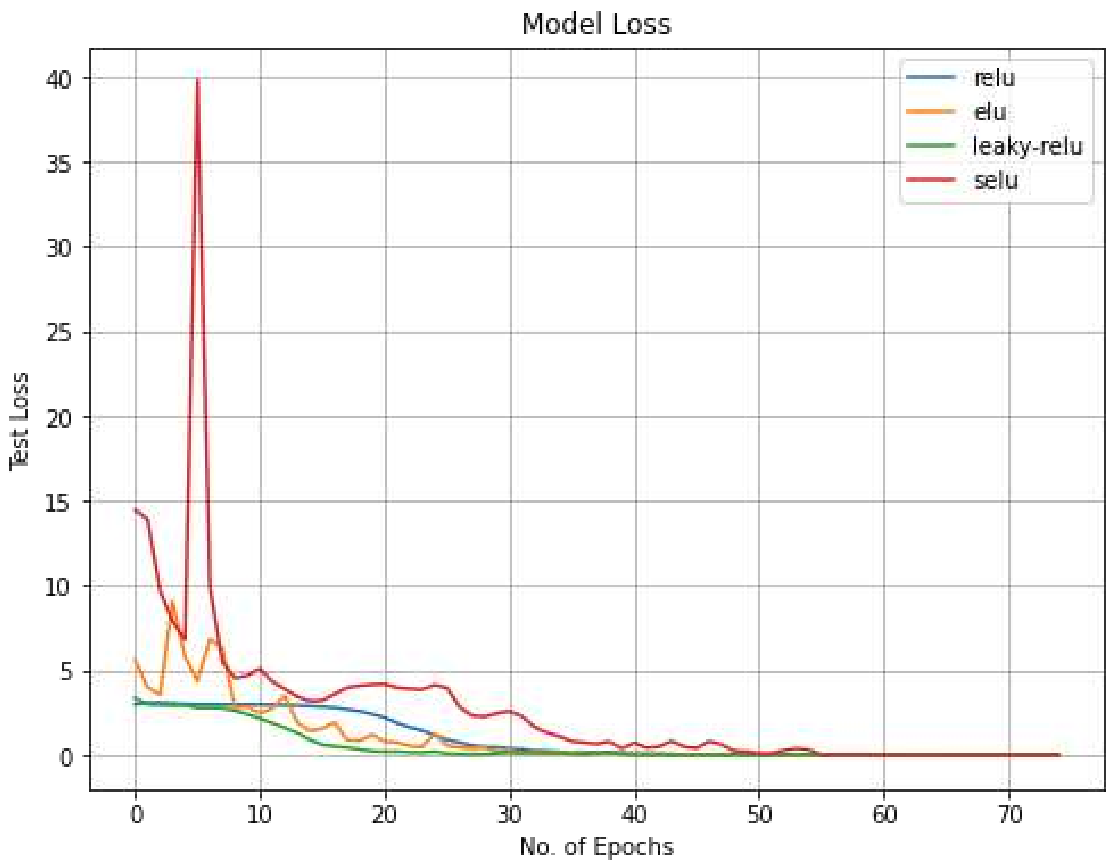
[33]:

```python
# Lets try to plot the Model accuracy and Model loss for each activation␣
 ↪function used above
# Just to make sure, we don't change the above data, so we store it in new␣
 ↪matrix

activation_list = activation_function[0:]
results_new = activation_results[0:]

def plot_results(activation_results,activation_functions_new =[]):

    plt.figure(figsize=(8,6))

    # Model accuracy values plot
    for activation_function in activation_results:
        plt.plot(activation_function.history['val_accuracy'])

    plt.title('Model accuracy')
    plt.ylabel('Test Accuracy')
    plt.xlabel('No. of Epochs')
    plt.legend(activation_functions_new)
    plt.grid()
```

```
    plt.show()

    # Model loss values plot

    plt.figure(figsize=(8,6))

    for activation_function in activation_results:
        plt.plot(activation_function.history['val_loss'])

    plt.title('Model Loss')
    plt.ylabel('Test Loss')
    plt.xlabel('No. of Epochs')
    plt.legend(activation_functions_new)
    plt.grid()
    plt.show()
```

[34]:
```
plot_results(results_new, activation_list)
```





Here it is seen that 'leaky-relu' and 'relu' both perform well with minimum loss at lower epochs as compared to other activation functions

Looking at the plots above all activation functions converge with minimum loss and high accuracy at training and validation set but 'leaky-relu' is able to converge for higher accuracy at lower epochs with minimum loss, so we choose 'leaky-relu' for final model training and plotting results.

[35]:
```
activation_func_final ='leaky-relu'

model_final = cnn_model(activation=activation_func_final,
                        dropout_rate=0.2,
                        optimizer=Adam(clipvalue=0.5)) #using 'adam' optimizer with␣
 ↪clipvalue of 0.5

history_final = model_final.fit(np.array(x_train), np.array(y_train),
                        batch_size=512,
                        epochs=75,
                        verbose=2,
                        validation_data=(np.array(x_valid),np.array(y_valid)))
```

```
Train on 216 samples, validate on 24 samples
```

```
Epoch 1/75
 - 5s - loss: 3.0276 - accuracy: 0.0556 - val_loss: 3.7636 - val_accuracy:
0.0000e+00
Epoch 2/75
 - 4s - loss: 5.7653 - accuracy: 0.0602 - val_loss: 4.2915 - val_accuracy:
0.0000e+00
Epoch 3/75
 - 4s - loss: 6.6994 - accuracy: 0.0648 - val_loss: 3.0085 - val_accuracy:
0.1250
Epoch 4/75
 - 4s - loss: 4.2439 - accuracy: 0.0602 - val_loss: 3.0025 - val_accuracy:
0.0000e+00
Epoch 5/75
 - 4s - loss: 3.1771 - accuracy: 0.0741 - val_loss: 3.1597 - val_accuracy:
0.0000e+00
Epoch 6/75
 - 4s - loss: 3.1659 - accuracy: 0.0741 - val_loss: 2.8108 - val_accuracy:
0.2083
Epoch 7/75
 - 5s - loss: 4.0284 - accuracy: 0.0880 - val_loss: 2.9313 - val_accuracy:
0.1250
Epoch 8/75
 - 4s - loss: 3.0218 - accuracy: 0.1019 - val_loss: 2.9868 - val_accuracy:
0.0833
Epoch 9/75
 - 4s - loss: 2.9041 - accuracy: 0.1204 - val_loss: 2.9900 - val_accuracy:
0.0000e+00
Epoch 10/75
 - 4s - loss: 2.9886 - accuracy: 0.0880 - val_loss: 2.9589 - val_accuracy:
0.0000e+00
Epoch 11/75
 - 4s - loss: 2.8874 - accuracy: 0.1250 - val_loss: 2.9278 - val_accuracy:
0.0000e+00
Epoch 12/75
 - 4s - loss: 2.8748 - accuracy: 0.0972 - val_loss: 2.8839 - val_accuracy:
0.0833
Epoch 13/75
 - 4s - loss: 2.8377 - accuracy: 0.1204 - val_loss: 2.8291 - val_accuracy:
0.2083
Epoch 14/75
 - 4s - loss: 2.7314 - accuracy: 0.2037 - val_loss: 2.7919 - val_accuracy:
0.2917
Epoch 15/75
 - 4s - loss: 2.7279 - accuracy: 0.2130 - val_loss: 2.7524 - val_accuracy:
0.3750
Epoch 16/75
 - 4s - loss: 2.6771 - accuracy: 0.2222 - val_loss: 2.6924 - val_accuracy:
0.2500
```

```
Epoch 17/75
 - 4s - loss: 2.6355 - accuracy: 0.2176 - val_loss: 2.6071 - val_accuracy:
0.3750
Epoch 18/75
 - 4s - loss: 2.4987 - accuracy: 0.3009 - val_loss: 2.5142 - val_accuracy:
0.3750
Epoch 19/75
 - 4s - loss: 2.3966 - accuracy: 0.3519 - val_loss: 2.4198 - val_accuracy:
0.3750
Epoch 20/75
 - 4s - loss: 2.3061 - accuracy: 0.3519 - val_loss: 2.3012 - val_accuracy:
0.3750
Epoch 21/75
 - 4s - loss: 2.1447 - accuracy: 0.3981 - val_loss: 2.0969 - val_accuracy:
0.6250
Epoch 22/75
 - 4s - loss: 2.0039 - accuracy: 0.4213 - val_loss: 1.8790 - val_accuracy:
0.6667
Epoch 23/75
 - 4s - loss: 1.8491 - accuracy: 0.4769 - val_loss: 1.7305 - val_accuracy:
0.7500
Epoch 24/75
 - 4s - loss: 1.7604 - accuracy: 0.4444 - val_loss: 1.5873 - val_accuracy:
0.8333
Epoch 25/75
 - 4s - loss: 1.5324 - accuracy: 0.5093 - val_loss: 1.3586 - val_accuracy:
0.8750
Epoch 26/75
 - 4s - loss: 1.4682 - accuracy: 0.5231 - val_loss: 1.2355 - val_accuracy:
0.8750
Epoch 27/75
 - 4s - loss: 1.2276 - accuracy: 0.6343 - val_loss: 0.9812 - val_accuracy:
0.9583
Epoch 28/75
 - 4s - loss: 1.1004 - accuracy: 0.6759 - val_loss: 0.8510 - val_accuracy:
0.8750
Epoch 29/75
 - 4s - loss: 0.9830 - accuracy: 0.6944 - val_loss: 0.9938 - val_accuracy:
0.7500
Epoch 30/75
 - 4s - loss: 0.9218 - accuracy: 0.7269 - val_loss: 0.6403 - val_accuracy:
0.8750
Epoch 31/75
 - 4s - loss: 0.7449 - accuracy: 0.8009 - val_loss: 0.5179 - val_accuracy:
0.9167
Epoch 32/75
 - 4s - loss: 0.5669 - accuracy: 0.8194 - val_loss: 0.4922 - val_accuracy:
0.9167
```

```
Epoch 33/75
 - 4s - loss: 0.5967 - accuracy: 0.8148 - val_loss: 0.2960 - val_accuracy:
0.9583
Epoch 34/75
 - 4s - loss: 0.4720 - accuracy: 0.8750 - val_loss: 0.2567 - val_accuracy:
0.9583
Epoch 35/75
 - 4s - loss: 0.4052 - accuracy: 0.8935 - val_loss: 0.2768 - val_accuracy:
0.9167
Epoch 36/75
 - 4s - loss: 0.2875 - accuracy: 0.9167 - val_loss: 0.2035 - val_accuracy:
1.0000
Epoch 37/75
 - 4s - loss: 0.2227 - accuracy: 0.9444 - val_loss: 0.1614 - val_accuracy:
0.9583
Epoch 38/75
 - 4s - loss: 0.2015 - accuracy: 0.9583 - val_loss: 0.1152 - val_accuracy:
1.0000
Epoch 39/75
 - 4s - loss: 0.1792 - accuracy: 0.9398 - val_loss: 0.1150 - val_accuracy:
1.0000
Epoch 40/75
 - 4s - loss: 0.1355 - accuracy: 0.9769 - val_loss: 0.0519 - val_accuracy:
1.0000
Epoch 41/75
 - 4s - loss: 0.1120 - accuracy: 0.9676 - val_loss: 0.0391 - val_accuracy:
1.0000
Epoch 42/75
 - 4s - loss: 0.0972 - accuracy: 0.9722 - val_loss: 0.0279 - val_accuracy:
1.0000
Epoch 43/75
 - 4s - loss: 0.1426 - accuracy: 0.9491 - val_loss: 0.0357 - val_accuracy:
1.0000
Epoch 44/75
 - 4s - loss: 0.0616 - accuracy: 0.9907 - val_loss: 0.0300 - val_accuracy:
1.0000
Epoch 45/75
 - 4s - loss: 0.0923 - accuracy: 0.9769 - val_loss: 0.0161 - val_accuracy:
1.0000
Epoch 46/75
 - 4s - loss: 0.0596 - accuracy: 0.9861 - val_loss: 0.0137 - val_accuracy:
1.0000
Epoch 47/75
 - 4s - loss: 0.0552 - accuracy: 0.9861 - val_loss: 0.0231 - val_accuracy:
1.0000
Epoch 48/75
 - 4s - loss: 0.0526 - accuracy: 0.9861 - val_loss: 0.0228 - val_accuracy:
1.0000
```

```
Epoch 49/75
 - 4s - loss: 0.0655 - accuracy: 0.9769 - val_loss: 0.0196 - val_accuracy:
1.0000
Epoch 50/75
 - 4s - loss: 0.0524 - accuracy: 0.9815 - val_loss: 0.0168 - val_accuracy:
1.0000
Epoch 51/75
 - 4s - loss: 0.0277 - accuracy: 0.9954 - val_loss: 0.0102 - val_accuracy:
1.0000
Epoch 52/75
 - 4s - loss: 0.0239 - accuracy: 0.9954 - val_loss: 0.0100 - val_accuracy:
1.0000
Epoch 53/75
 - 4s - loss: 0.0362 - accuracy: 0.9861 - val_loss: 0.0093 - val_accuracy:
1.0000
Epoch 54/75
 - 4s - loss: 0.0146 - accuracy: 1.0000 - val_loss: 0.0053 - val_accuracy:
1.0000
Epoch 55/75
 - 4s - loss: 0.0213 - accuracy: 0.9954 - val_loss: 0.0033 - val_accuracy:
1.0000
Epoch 56/75
 - 4s - loss: 0.0211 - accuracy: 0.9954 - val_loss: 0.0033 - val_accuracy:
1.0000
Epoch 57/75
 - 4s - loss: 0.0216 - accuracy: 1.0000 - val_loss: 0.0014 - val_accuracy:
1.0000
Epoch 58/75
 - 4s - loss: 0.0125 - accuracy: 1.0000 - val_loss: 0.0013 - val_accuracy:
1.0000
Epoch 59/75
 - 4s - loss: 0.0185 - accuracy: 0.9907 - val_loss: 0.0014 - val_accuracy:
1.0000
Epoch 60/75
 - 4s - loss: 0.0116 - accuracy: 1.0000 - val_loss: 0.0017 - val_accuracy:
1.0000
Epoch 61/75
 - 4s - loss: 0.0044 - accuracy: 1.0000 - val_loss: 0.0020 - val_accuracy:
1.0000
Epoch 62/75
 - 4s - loss: 0.0218 - accuracy: 0.9907 - val_loss: 0.0040 - val_accuracy:
1.0000
Epoch 63/75
 - 4s - loss: 0.0128 - accuracy: 1.0000 - val_loss: 0.0062 - val_accuracy:
1.0000
Epoch 64/75
 - 4s - loss: 0.0187 - accuracy: 0.9861 - val_loss: 0.0118 - val_accuracy:
1.0000
```

```
Epoch 65/75
 - 4s - loss: 0.0236 - accuracy: 0.9954 - val_loss: 0.0061 - val_accuracy:
1.0000
Epoch 66/75
 - 4s - loss: 0.0070 - accuracy: 1.0000 - val_loss: 0.0036 - val_accuracy:
1.0000
Epoch 67/75
 - 4s - loss: 0.0112 - accuracy: 1.0000 - val_loss: 0.0021 - val_accuracy:
1.0000
Epoch 68/75
 - 4s - loss: 0.0130 - accuracy: 0.9954 - val_loss: 0.0029 - val_accuracy:
1.0000
Epoch 69/75
 - 4s - loss: 0.0101 - accuracy: 1.0000 - val_loss: 0.0054 - val_accuracy:
1.0000
Epoch 70/75
 - 4s - loss: 0.0153 - accuracy: 0.9907 - val_loss: 0.0135 - val_accuracy:
1.0000
Epoch 71/75
 - 4s - loss: 0.0153 - accuracy: 0.9954 - val_loss: 0.0131 - val_accuracy:
1.0000
Epoch 72/75
 - 4s - loss: 0.0066 - accuracy: 1.0000 - val_loss: 0.0095 - val_accuracy:
1.0000
Epoch 73/75
 - 4s - loss: 0.0149 - accuracy: 0.9954 - val_loss: 0.0050 - val_accuracy:
1.0000
Epoch 74/75
 - 4s - loss: 0.0171 - accuracy: 0.9907 - val_loss: 0.0022 - val_accuracy:
1.0000
Epoch 75/75
 - 4s - loss: 0.0088 - accuracy: 1.0000 - val_loss: 0.0014 - val_accuracy:
1.0000
```

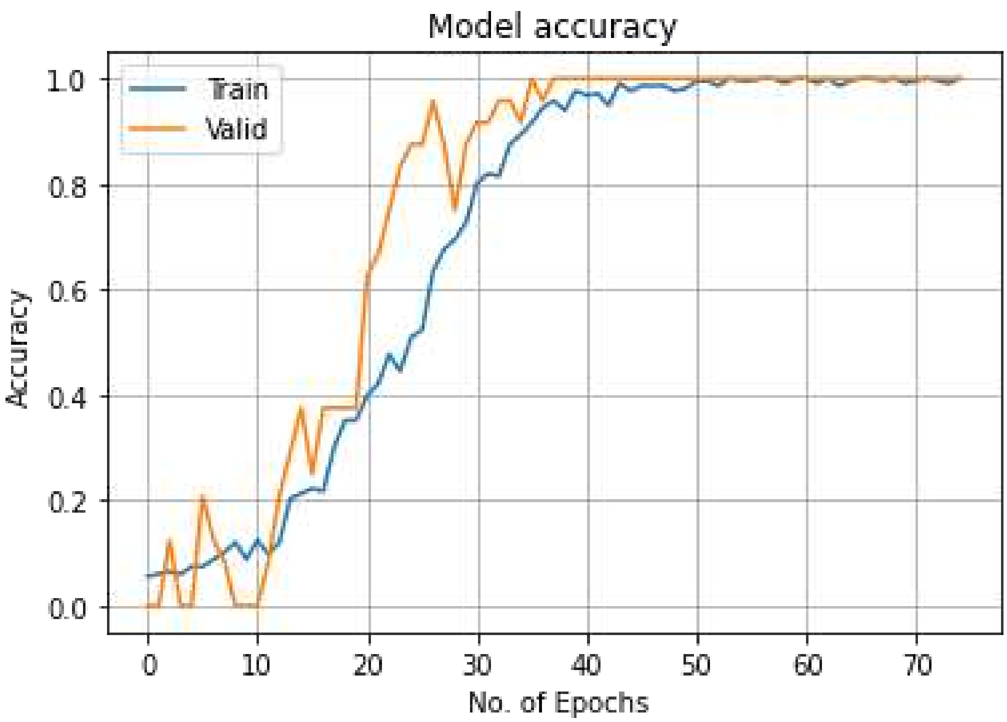
[36]:
```
result_score = model_final.evaluate(np.array(x_test),np.array(y_test),verbose=0)

print('Test Loss {:.4f}'.format(result_score[0]))
print('Test Accuracy {:.4f}'.format(result_score[1]))
```

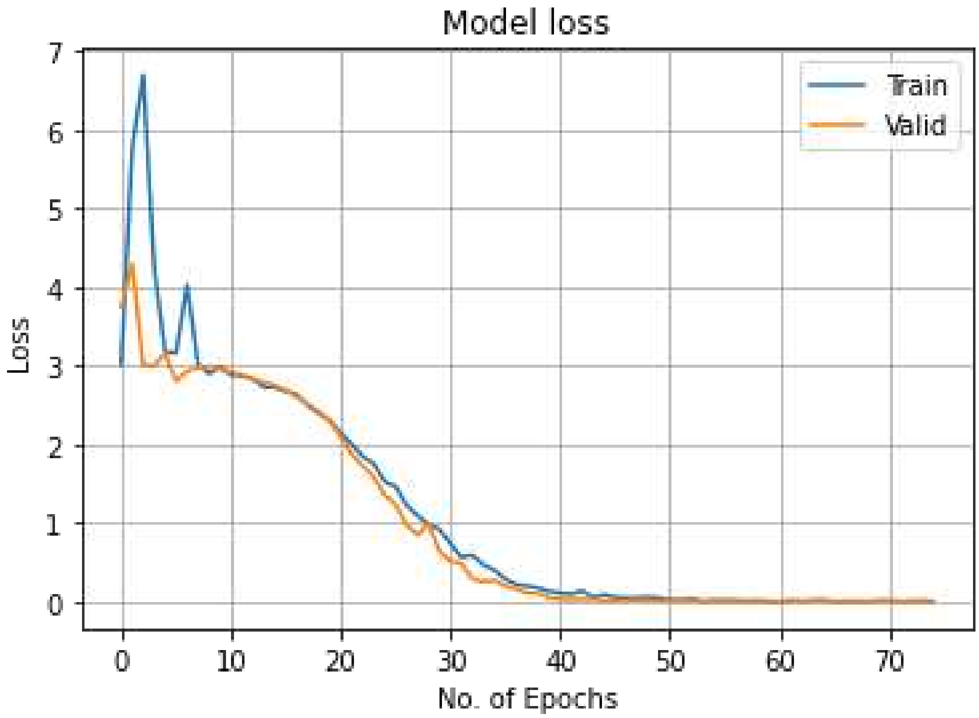
```
Test Loss 0.2217
Test Accuracy 0.9500
```

[37]:
```
# Data in history

print(history_final.history.keys())

# Plotting Accuracy for final model
```

```
plt.plot(history_final.history['accuracy'])
plt.plot(history_final.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel(' No. of Epochs')
plt.legend(['Train', 'Valid'])
plt.grid()
plt.show()

# Plotting Loss for Final Model
plt.plot(history_final.history['loss'])
plt.plot(history_final.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('No. of Epochs')
plt.legend(['Train', 'Valid'])
plt.grid()
plt.show()
```

```
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```





Conclusion

Here in this project we analyzed ORL faces images (train and test sets were given). We used CNN method to build the model and train it.

The analysis for different activation functions is fisrt observed to find that 'leaky-relu' activation function is one of the activation functions that can be used for out final model

The model training is done using x_train and y_train with validation data as x_valid and y_valid. owever for evaluating model, we use x_test and y_test which gives us loss ~0.2217 with an accuracy of 95.00%

[ ]:

```

```