Handwritten Greek characters recognition

Shamika Pradhan

Submitted for the Degree of Master of Science in

Artificial Intelligence

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

December 13, 2021

Declaration

This report has been prepared based on my work.
Where other published and unpublished source materials have been used, those have been acknowledged.

Word Count: 13437

Student Name: **Shamika Nandan Pradhan**

Date of Submission: 13 December 2021

Signature: Shamika

Acknowledgments

I would like to express my gratitude to my mentor professor Nicolo Colombo who listened to my problems and nervousness towards this project and made sure that I complete it with confidence and also, I would like to thank my Machine Learning Professor Vladimir Vovk and Deep learning Professor Dr. Chris Watkins for making many concepts clear during the course period which helped me a lot to complete this project. I would also like to thank Ms. Eva Dann for helping me out with the referencing rules and solving my doubts regarding the same.

And lastly, I would like to thank my parents and friends for supporting me and helping me throughout my master's course.

Abstract

Greek characters are used to write the Greek language which is the official script for Greece, Cyprus, and European Union. Greek characters are one of the old script characters which are "*today used for writing modern Greek and symbols in mathematics and science*" (Greece.com, 2021). Processing and working with handwritten Greek characters is an interesting field of study as a "*Greek text search facility is far from perfect*" (Rhul.ac.uk, 2021). This project aims at running and comparing three models: *1)ResNet50 2)CNN and 3)RNN* on *High-Quality Greek character dataset* and *Low-Quality Greek character dataset* to see if the quality of the dataset given for training the model plays an important role to get good accuracy to while testing the model and predicting the Character, making it interesting to compare the models and their accuracies based on the quality of the training dataset, furthermore the comparison is visualized with the help of Bar plots and other Graphs making it easy to understand and compare differences between the accuracies and errors of each model. There are 24 Greek characters starting from alpha(α) to omega(ω). The predicted Labels are visualized with the help of Images and bar plots highlighting the predicted label. The code is written in *Python* language using *Jupyter notebook* which is easy to understand and use for further research and experiments with the dataset and models.

Contents:

1. Introduction

English characters are easy to understand and differentiate. There are many English text recognition software's but there is no proper text recognition software for handwritten Greek characters as the characters are similar looking.

This project will help us understand the overall performance of each model and which is the best when working with a handwritten Greek dataset whether it is a low-quality or a high-quality dataset.

This project will also help us understand whether the quality of the dataset, partition of the dataset into train, validation, and test play an important role while training and evaluating the model and predicting the results.

Each model is given the same dataset with the same partition of training and validation to make a fair evaluation.

Throughout the project, TensorFlow and Keras are the widely used libraries to import inbuilt models and layers.

Both the datasets have 24 classes as α(alpha), β(beta), γ(gamma), δ(delta), ε(epsilon), ζ(zeta), η(eta), θ(theta), ι(iota), κ(kappa), λ(lambda), μ(mu), ν(nu), ξ(xi), o(omicron), π(pi), ϱ(rho), σ(sigma), τ(tau), υ(upsilon), φ(phi), χ(chi), ψ(psi), ω(omega). As the training and validation are split into 60% and 40% respectively making it a fair distribution while feeding the model and then the model is evaluated on the validation dataset to change or add any missing parameters to the optimizer making it a tuned model before running it on the test dataset.

The models are evaluated based on maximum accuracy and minimum loss, which are then visualized with the help of bar plots and graphs. Furthermore, the predictions help us understand how good really the obtained accuracy is and how truly can it predict any test image.

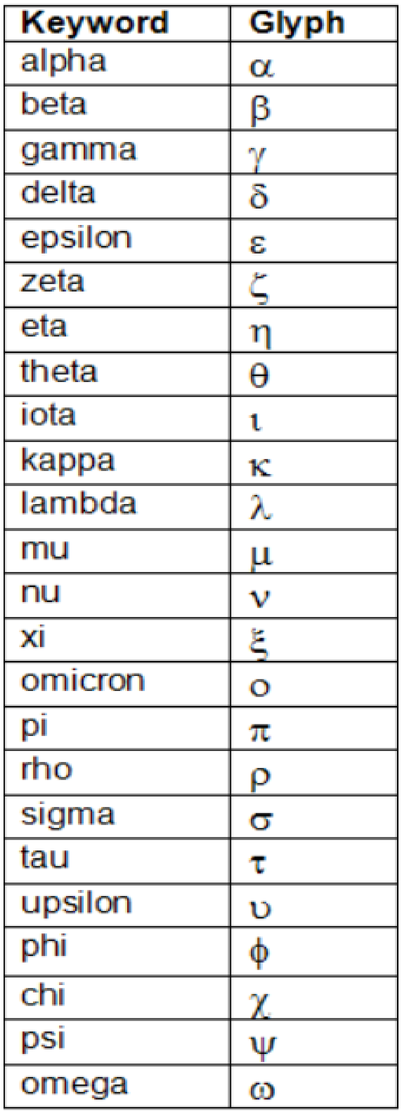
This section gives an overview of what the project is exactly about and gives an understanding of what are the 24 Greek letters and what are their symbols.

2. Background Research

2.1 Greek characters

The first thing I did when started doing this project was to get a fair knowledge of the dataset which I will be working on. Greek characters were an unknown concept for me as I only knew some of the characters used for scientific notations.

There are 24 Greek letters and when combined with signs and symbols they become 270 Greek character letters. The recognition system for Greek letters was very hard to find as the letters are very tricky to understand. The Greek alphabet is used from the late 900 BC in the Greek language making it an ancient script.



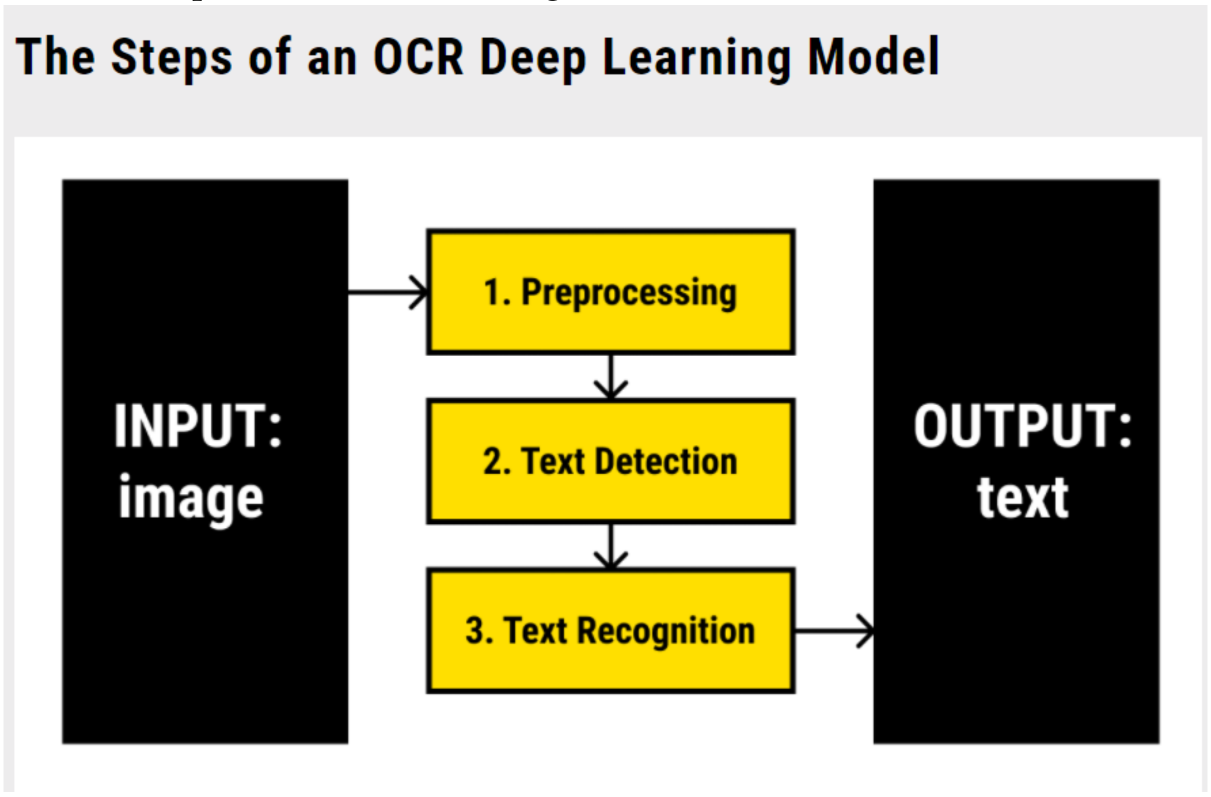
Keyword	Glyph
alpha	α
beta	β
gamma	γ
delta	δ
epsilon	ε
zeta	ζ
eta	η
theta	θ
iota	ι
kappa	κ
lambda	λ
mu	μ
nu	ν
xi	ξ
omicron	ο
pi	π
rho	ρ
sigma	σ
tau	τ
upsilon	υ
phi	φ
chi	χ
psi	ψ
omega	ω

(SAS, 2007)

The above image shows the Greek character names and their respective symbols. As you can see digital Greek characters are very hard to understand and also to differentiate that's why the main challenge ahead was to work with handwritten Greek Characters. This was the base for me to find a perfect database to feed my models to get good accuracy.

The image also shows the similarity between zeta and xi making it difficult for the models to understand a significant difference between them.

2.2 OCR (Optical Character Recognition)

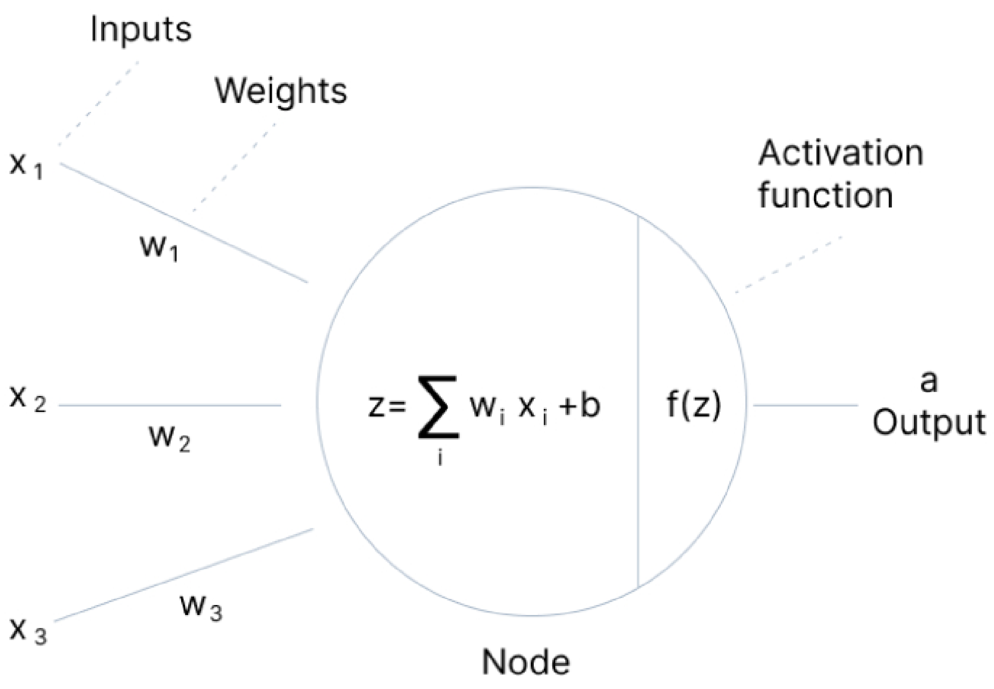


(Sydorenko, 2021)

Optical Character Recognition is used to recognize text from an image, it can be a scanned document or photo basically a digital image. Optical Character Recognition technology is used to recognize any text hidden in the image or in the form of an image and convert it into machine understandable text. Optical Character Recognition has many important applications from digitizing the newspaper to word processors like Microsoft Word and Google Docs. (Learning, G 2020)

2.3 Activation Functions

Activation Functions in a neural network are responsible for transforming the summed up weights obtained from the other nodes to the activation of the output node or any other input node.



(Baheti, P and V7 labs 2021)

In the above image the node which has the equation z = $\sum_i wixi + b$, which means the summation of all the inputs and weights in addition to the bias, further f(z) is the activation function it can be ReLu, Sigmoid, SoftMax or any other activation function which gives us an output a.

2.3.1 ReLu (Rectified Linear Unit)

ReLu is a Rectified linear activation function and is the most commonly used activation function as it switches off the node if the function gets “0” or any “negative values” as its input or remains on if the function gets any “positive values” as its input.

In the above activation function, f(z) can be written as f(z) = max(0,z) where if the value of z is greater than 0 it will be returned as an output because of the max function, and the node will remain activated but if z is a negative value max function will return 0 as the maximum number and the node will be set to off.

2.3.2 Sigmoid

The Sigmoid function is also an activation function used to convert the input into probability which can also be stated as mapping the values into very small numbers that are between 0 to 1 or -1 to 1. In the above activation function f(z) can be written as f(z) = $\frac{1}{1+e^{-z}}$ compared to the Rectified linear activation function sigmoid function is slow and possess the problem of vanishing gradients. Choosing ReLu over Sigmoid can eliminate the vanishing gradients problem.

2.3.3 SoftMax

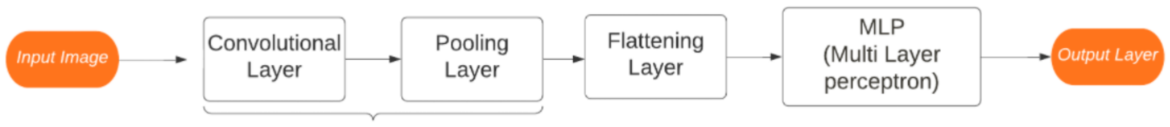
The SoftMax function is an activation function that can be said as an improved version of the sigmoid where the vanishing gradients problem is solved. It takes negative, positive, zero, or any number greater than zero and maps it to a smaller probability lying in the range of 0 to 1.

If the input is a small number, it easily maps them to a smaller probability between 0 and 1, and when the input is a larger number into a large probability which still lies between 0 and 1. It can be used for multi class classification in our case is the main aspect of this project.

2.4 Image Recognition and Classification

Image recognition means feeding images or we can say image dataset to the neural network to process it to get a labeled output. This is an important procedure in my project as the data which is fed to the neural network is of handwritten Greek Character images.

The most popular neural network used for image classification is the Convolutional neural network. Image recognition means identifying the input image and classifying it into, what type is the input image exactly for which labels are used.



(JACKOWSKI and JACKOWSKI, 2021)

The input image consists of pixels. In this project, I have used a Grayscale image (black and white) which is displayed as a 2d array that falls in the range of values from 0 to 255. One can also use an RGB image (coloured image) which is transformed into a 3D array which means each layer represents colours.

The Convolutional layer helps to detect some features of the input image and its output is called a feature map which is an input to the ReLu rectifier to make them separable and easy to use. The output of these layers is then given as an input to the max pooling layer then the out is flattened in the flattening layer to make the matrix acceptable in the Artificial Neural Network. In the end, the Fully connected layer is responsible for the classification of the image which was given as the input. When we say visualization of the classifier the first thing that pops in the head is the confusion matrix.

2.5 Predictions and Confidence measures

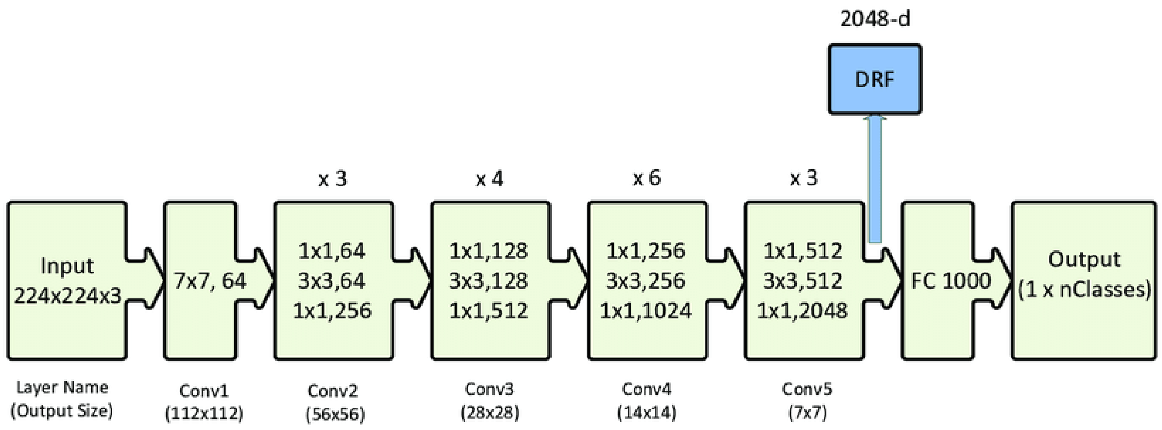
The confidence measure of the prediction is how confident the model is about the accuracy and the scores it is getting. The most common confidence score is **between 0 and 1** this is easy to understand. The other confidence score that can be used is **between 0 to + ∞ or –∞ to +∞**. The next one is pretty interesting and an easy way to understand because it tells whether the confidence score is **"low", "medium" or "high"**. Now let us talk about true positives, true negatives, false positives, and false negatives these metrics refers to binary classification problems. True positives mean "yes", and correct, True negatives mean "no" and correct, false positives mean "yes" and wrong, and lastly, false negatives mean "no" and wrong. The accuracy is also one of the most used confidence measures metrics for example one can say there is a 91% accuracy to this model and the other 9% is the error or the loss. Accuracy is used when there is no significant trade-off between True positives and True negatives. **Accuracy** does depend on True positives, True negatives, False positives, and False negatives as the formula for accuracy is equal to **(true positive + true negative)/(true positive + true negative + false positive + false negative).** Then comes **Recall and Precision** where recall tells the sensitivity of the model it is a percentage where the data points predicting "yes" from the test dataset are divided by the data points actually holding the true values "yes", the formula for the recall is equal to **(true positive)/(true positives + false negatives)**. Precision means computing the positive predicted values of the model it also tells that how much we should trust the model when it predicts "yes" or "true". Precision is basically a proportion of the properly predicted "true" values vs the guessed predicted "true" values, it gives confidence about the model we are building, to measure the model precision we can compute the percentage of real "yes" to all the predicted "yes", the formula for the precision is equal to **(true positive)/(true positives + false positives)**. (Grandperrin, 2021)

2.6 Models

2.6.1 ResNet50:

ResNet50 is one of the models used in this project. It is a Convolutional Neural Network that is 50 layers deep and considered much better when dealing with image datasets. ResNet50 is a deep residual network. It is a type of ResNet model which is 48 layers deep with 1 Maxpooling layer and 1 Averagepooling layer. ResNet50 is considered to be a gift to the machine learning and deep learning field as its framework can easily be used to train complex and deep neural networks and still achieve great results with great performance.

ResNet50 is considered best when dealing with image datasets but due to its ability to easily deal with complex neural networks can also give good accuracy for non-image type tasks.

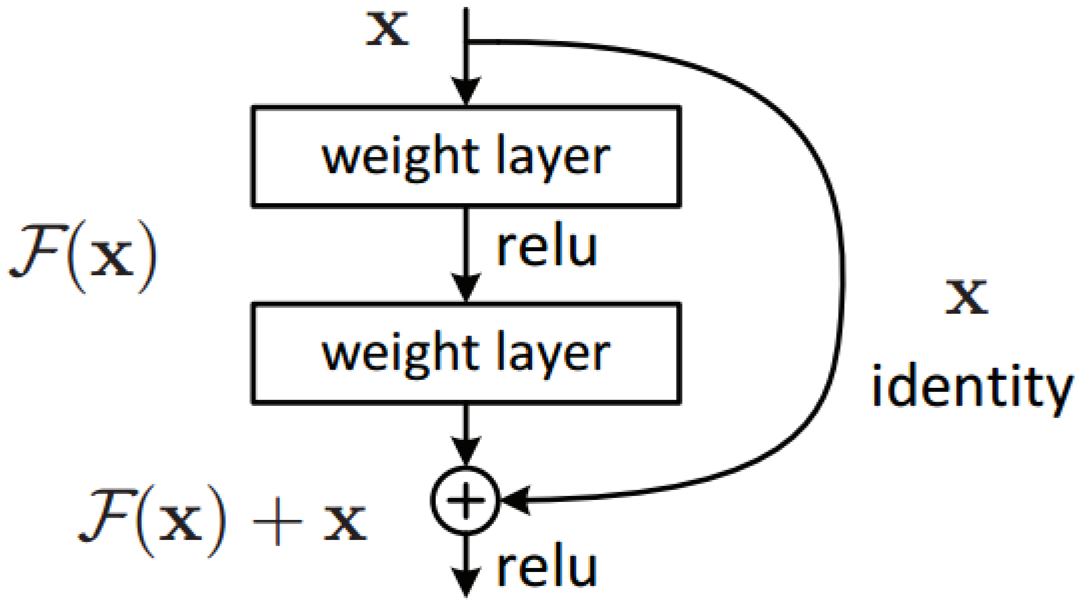


(Mahmood *et al.*, 2020)

The above figure gives a fair understanding of what and how exactly is the Architecture of ResNet50 where the input is given of size 224X224X3 where 224 is the filter size and 3 is the number of channels. FC 1000 is basically a Fully connected layer with 1000 neurons. x3, x4, x6, x3 written on the top of the Convolutional units denotes how many times does the specific unit repeats. For example, the Conv2 unit repeats 3 times.

It is globally accepted that training of the model becomes more difficult as more layers are added but with the introduction of ResNet by "Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in their 2015 computer vision research paper on computer vision titled 'Deep Residual Learning for Image Recognition." (viso.ai, 2021) most of the issues faced earlier were surpassed making ResNet world-class in ImageNet. ResNet has many types and ResNet50 is one of them the "50" denotes the number of layers present in the Residual Network.

ResNet was made to tackle many issues where the main issue was vanishing gradients caused by adding a lot of layers to the neural network which led to saturation of the accuracy giving over-fitted results on the testing data. ResNet uses the concept of Residual block which helps with the saturation of the accuracy and improves it.


x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

(He et al., 2015)

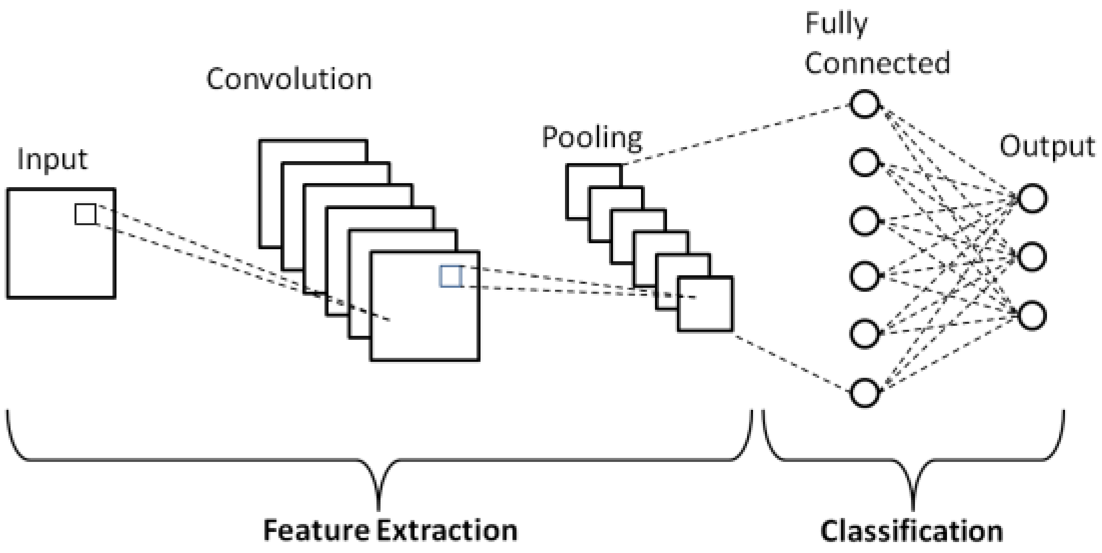
The above image is of a Single residual block. As the name suggests it means the residue F(x). Where F(x) = Output – Input i.e. H(x) – x

When rearranged we get H(x) = F(x) + x, basically in the residual block the layers are trying to learn the residual between the layers. ResNet has Skip Connections meaning adding information from the previous layers to the next layer which eliminates the issue of vanishing gradients In Convolutional Neural networks and with the combination with Residual network ResNet can efficiently improve the accuracy in deep neural networks while cutting down the error percentage.

2.6.2 CNN:

As we already know ResNet is the new and improved version of CNN but without knowing what exactly CNN is, no one would have known the concept of ResNet50.

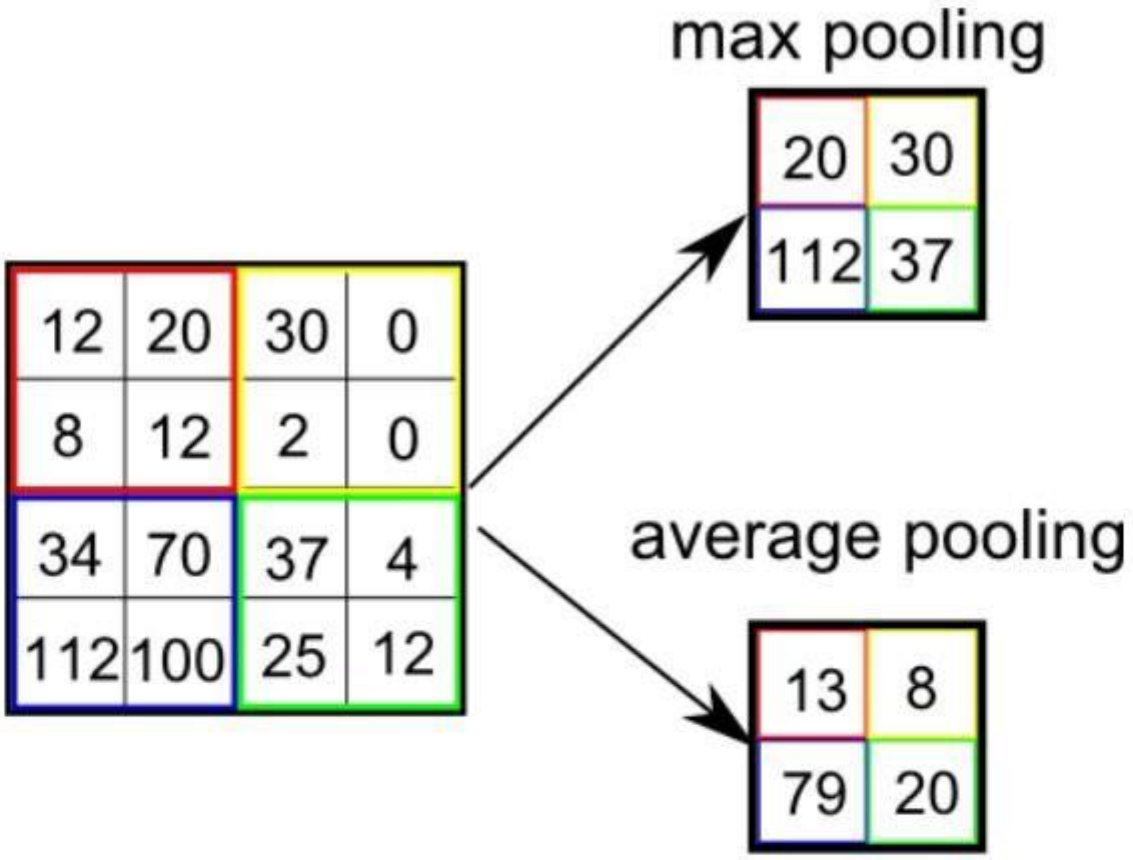
CNN is Convolutional Neural Network which is a deep learning algorithm. It takes images as input assigns them importance, we can also call them weights and biases then it processes them to classify based on the labels and evaluates that is one image given as the test input can be differentiated from the other test image.



(Balaji, 2020)

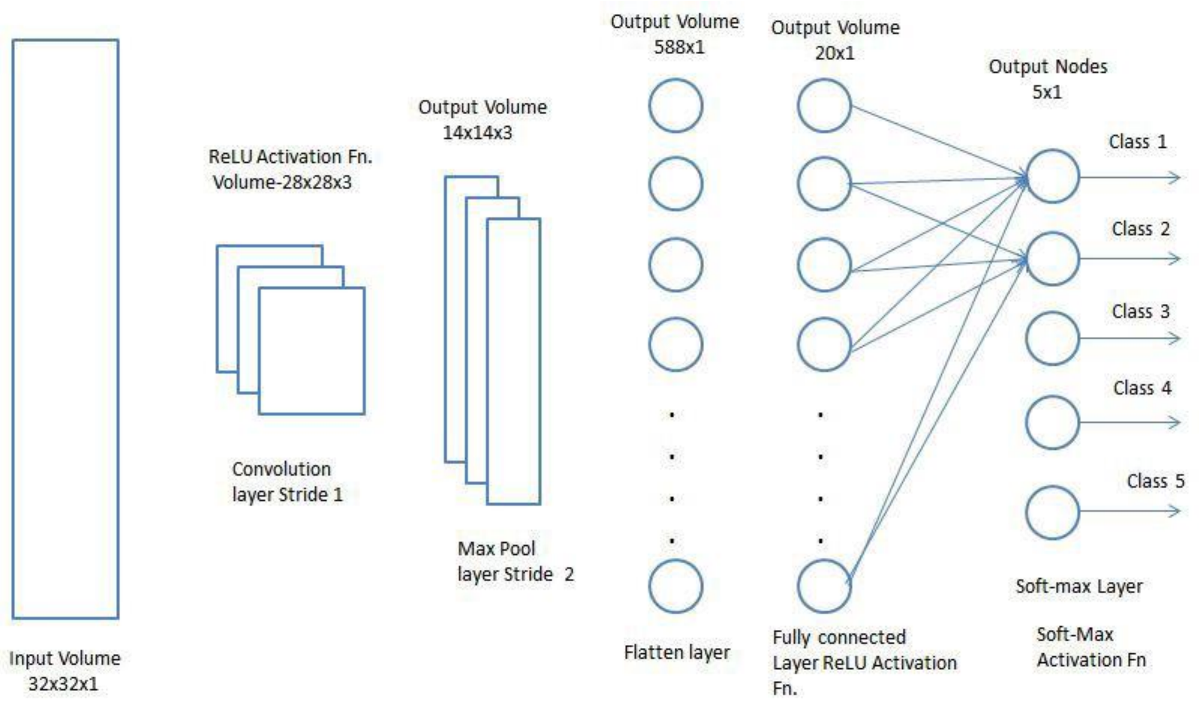
The above image describes the outline of a Convolutional Neural network. The architecture of the Convolutional Neural Network is analogous to that of the neuron structure of the human brain as the main concept of the NN's is to replicate the thinking and learning process of Humans.

Convolutional Neural Network is a Feed-Forward neural network meaning there is only forward propagation and no backward propagation. Normal Neural networks when applied to flatten to covert for example (3X3) to (9X1) sometimes give bad accuracies when performing prediction of classes with complex images and having dependency of pixels throughout, But Convolutional Neural Network successfully captures the dependencies that the normal neural network can't with the help of relevant and important filters making it a perfect base for ResNet. Reduction of the images into the form where they are made easier to process without losing the quality or any features. The number of parameters and the reusability of weights are reduced efficiently to capture the Temporal and Spatial dependencies.



(Saha, 2018)

As previously discussed, reductions of Spatial dependencies in the Convolutional layers, the Pooling layer is also responsible for the reduction of Spatial dependencies by reducing the computational energy necessary for the processing of the data. There are two types of Pooling Max Pooling and Average Pooling layers where Max pooling layer is responsible for the Suppression of the noise produced and the Average pooling layer is responsible for the reduction of dimensionality. But interestingly Average pooling performs dimensionality reduction just as the noise suppression mechanism makes it less important than the Max Pooling layer.



(Saha, 2018)

Then comes the classification Fully connected layer where the flattened image is given as the input to a feed-forward neural network and then is classified with the help of SoftMax function.

SoftMax is an activation function where,

Formula

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

σ = softmax

$\vec{z}$ = input vector

e^{z_i} = standard exponential function for input vector

K = number of classes in the multi-class classifier

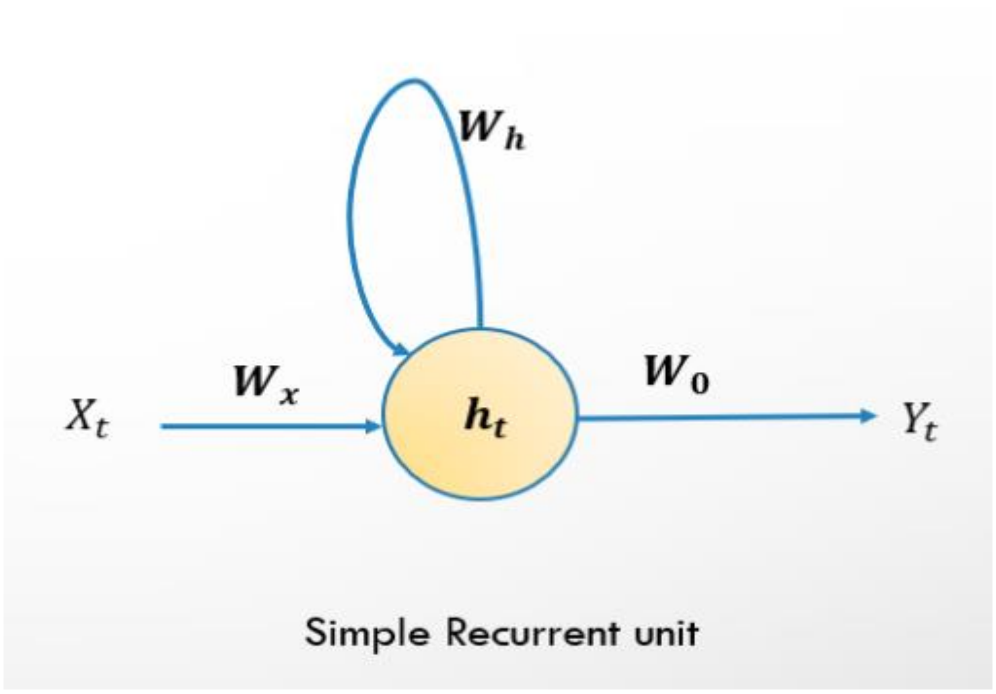
e^{z_j} = standard exponential function for output vector

e^{z_j} = standard exponential function for output vector

"SoftMax function turns vector of k real values into K real values that sum to 1" (Wood, 2019).

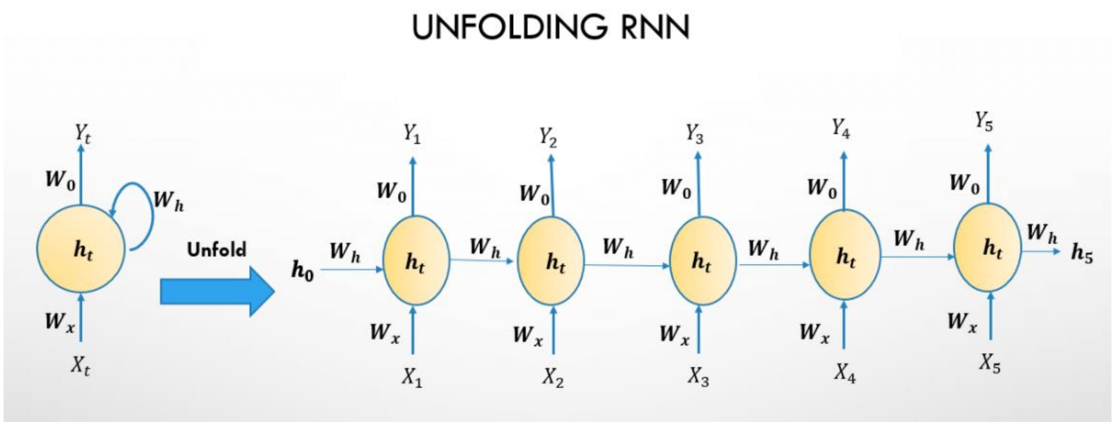
2.6.3 RNN:

Unlike ResNet50 and CNN which we saw earlier, RNN is a Recurrent Neural Network, and is basically used for sentiment classification, sequence classification, etc. It is a class of ANN (Artificial Neural Network) in which the connections present between the nodes produce a directed graph along a sequence of happening in space-time which is also referred to as temporal Sequence.



(Anwla, 2020)

The above image is an architectural representation of a Simple Recurrent Neural Network with weights where, Xt is the input Wx is the weight added and ht is the function from which the input and the weight must go through, Yt represents the output. The function can be anyone but are usually Sigmoid, tanh or ReLu. Wh represents the feedback loop which will be easier to understand with the help of the below image for unfolding the loops.



(Anwla, 2020)

Let us assume that the length of the sequence is 5 then after unfolding the loop we can see that it basically repeats 5 times in other words we can say it has 5 hidden layers.

A Recurrent Neural network is not like the traditional neural network which uses different parameters for different layers as a Recurrent neural network provides the same weights and biases to the layers converting the independent activations to dependent activations, in other words in recurrent neural network same tasks are performed on the same layers but with different inputs. This helps in the reduction of the parameters to be learned to make it easy to use. Recurrent Neural Networks are less complicated when compared to ResNet or CNN which deals with deep neural networks.

A real-life application of RNN is sentiment analysis which is widely used in the entertainment industry as well as the business industry which helps the organization know their positive and negative points extracted from a large database of reviews.

2.7 Python Libraries

The Python libraries contain built-in modules that are responsible to provide access to the system functions for example opening a file. These libraries are written in C language. Python libraries are life savers as they eliminate the need for writing codes from scratch as they come with built-in functions. In this project, I have used TensorFlow and Keras as they are the most important libraries for the models used. They contain built-in Neural network models such as ResNet50, CNN, and SimpleRNN. Which will be see in detail in the Experiments section.

2.7.1 TensorFlow, Keras, and TensorFlow VS PyTorch

Unlike TensorFlow, Keras is a built-in library in python thus more user friendly than TensorFlow. They provide higher-level API for Building models and training them easily. It is a free open-source library that is widely used in the Machine learning and Artificial Intelligence sector. Most of the TensorFlow is written in High-level C++ and CUDA.

TensorFlow is easy to import with the help of the pip command through the command prompt. Which will be seen in detail in the Experiments section. Personally, I had an option between PyTorch and TensorFlow, but Pytorch was more object-oriented and TensorFlow comes with different other options from which we can choose making it user friendly and a better option.

PyTorch is easy to use when dealing with light work datasets whereas TensorFlow is much more complex and better for model production with good stability. I have done many deep learning assignments with the help of PyTorch but every time I ran into some of the other versions and library errors making it the less likely to use the library.

Even when compared to Keras, Pytorch is still undesirable even though it is faster than Keras the quality is still less as we discussed earlier unlike Pytorch Keras provides higher-level API. TensorFlow supports many classifications as well as regression algorithm.

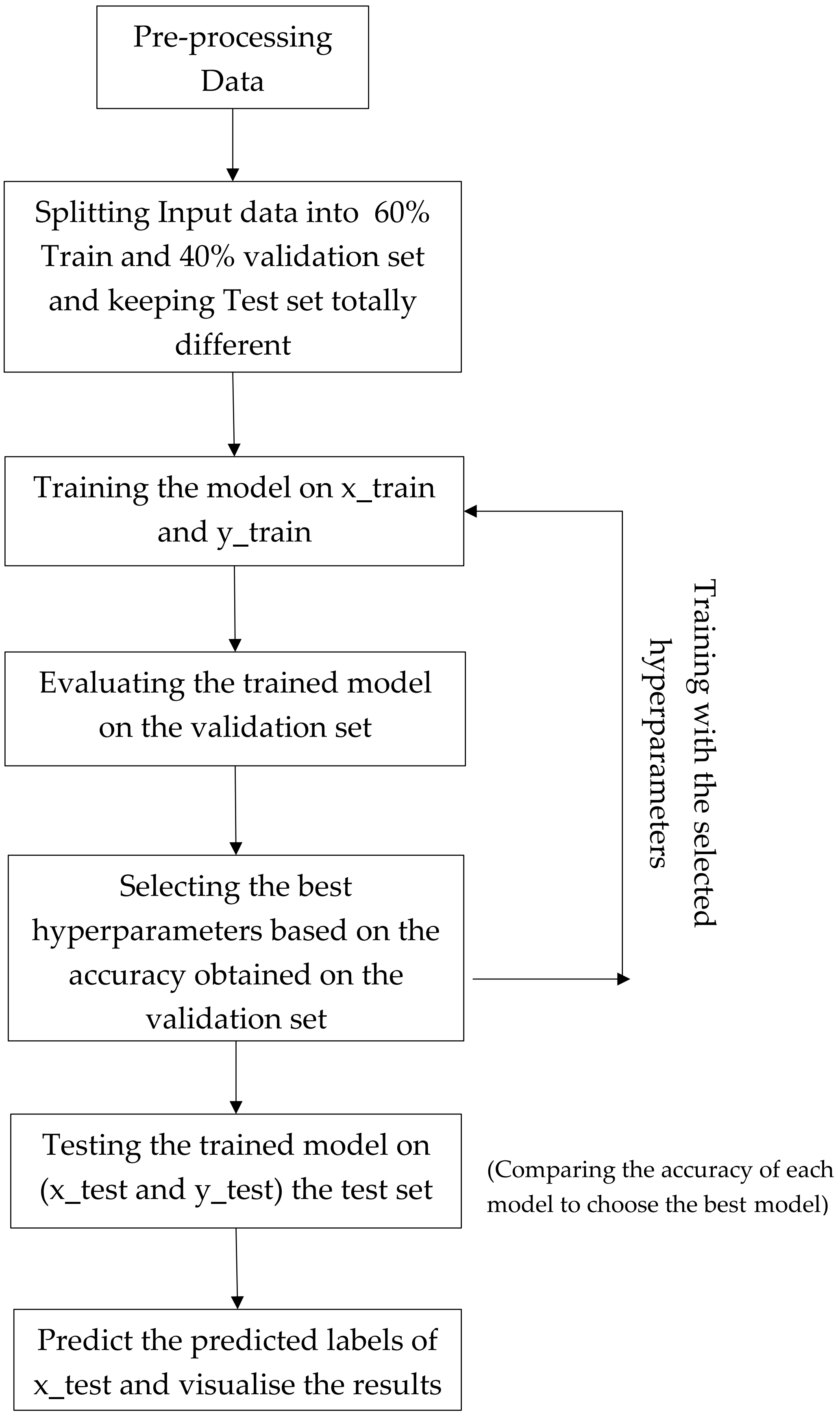
3. Experiment

3.1 Aim

Implementing several learning algorithms to recognize the ancient Greek characters. A recent Greek database with ground-truth information will be provided for training and testing (Rhul.ac.uk, 2021). The suggested extension was to predict the Greek characters with confidence measures.

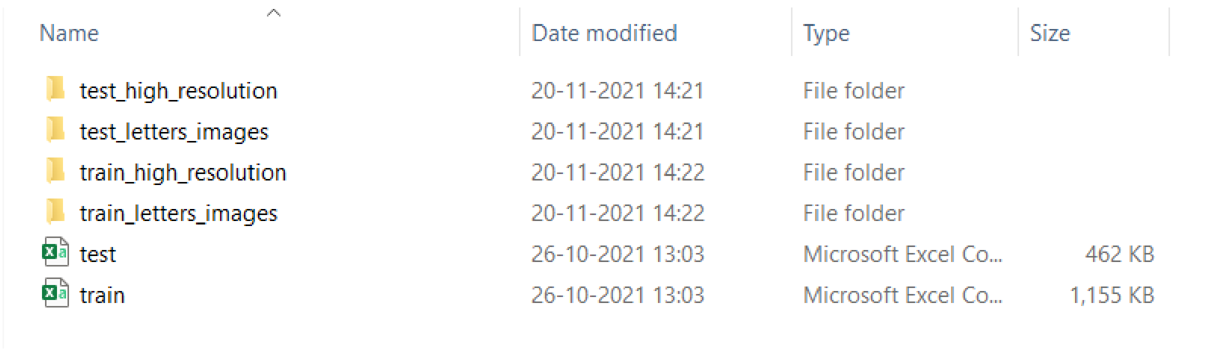
1. Proof of concept program: identify one machine learning task suitable for characters recognition and implement one learning algorithm for the chosen task.
2. Report: An overview of the datasets. Statistics and numbers involved.
3. Report: A report on machine learning tasks for character recognition.
4. Survey of machine learning techniques used for characters recognition.
5. The developed machine learning software for Greek characters recognition.
6. The software engineering process involved in generating your software.

3.2 Flow Chart


Pre-processing Data
Splitting Input data into 60% Train and 40% validation set and keeping Test set totally different
Training the model on x_train and y_train
Evaluating the trained model on the validation set
Selecting the best hyperparameters based on the accuracy obtained on the validation set
Training with the selected hyperparameters
Testing the trained model on (x_test and y_test) the test set
(Comparing the accuracy of each model to choose the best model)
Predict the predicted labels of x_test and visualise the results

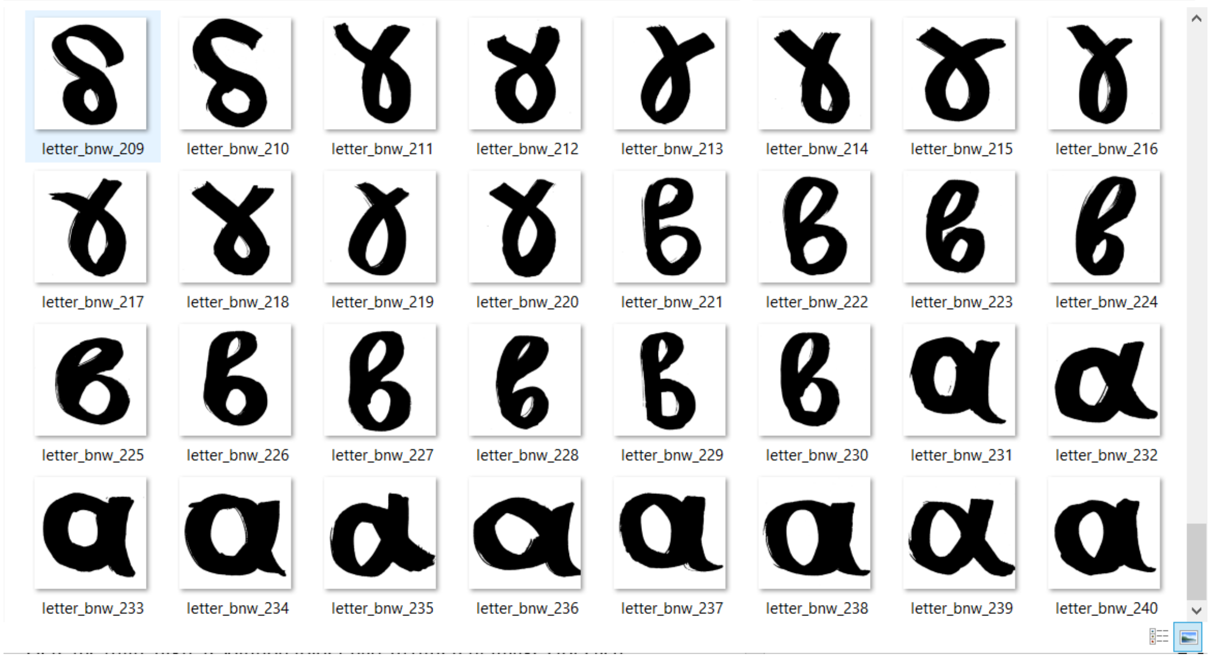
3.3 Datasets

The first task for this project was to collect data on handwritten Greek Characters. I got a dataset from **Kaggle (Kontolati, 2020)** which had High- Resolution train Greek letters, Low- Resolution train Greek letters, High- Resolution test Greek letters, and Low-Resolution train Greek letters and two CSV files each for test and train.

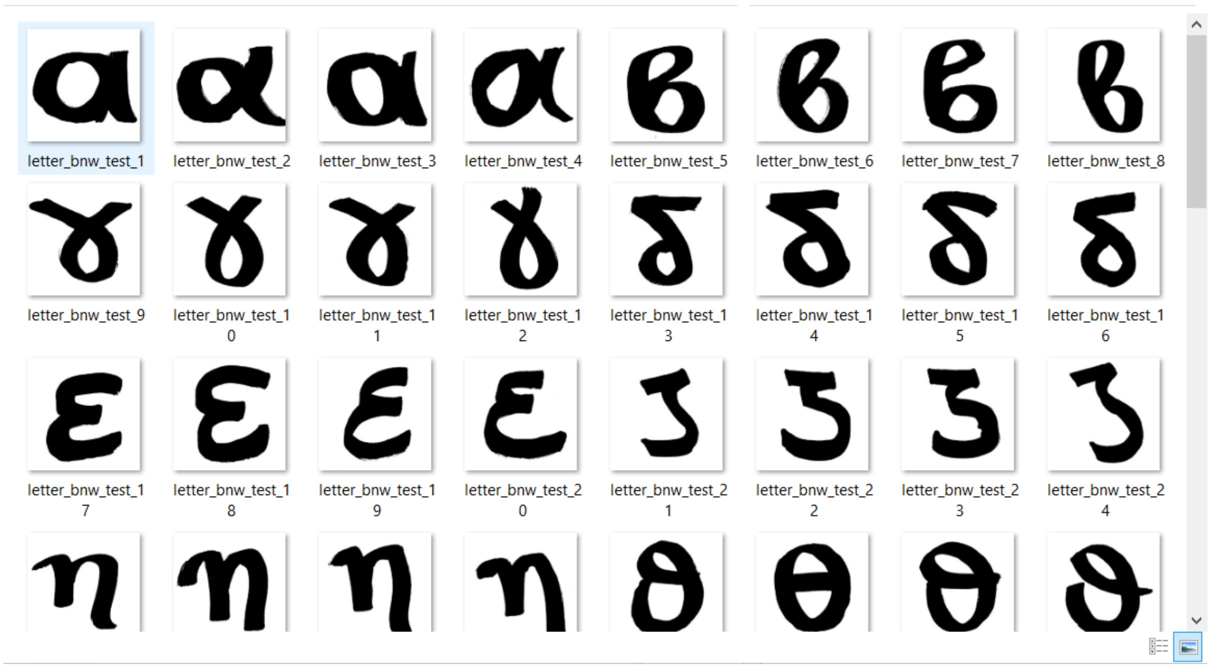


Name	Date modified	Type	Size
test_high_resolution	20-11-2021 14:21	File folder	
test_letters_images	20-11-2021 14:21	File folder	
train_high_resolution	20-11-2021 14:22	File folder	
train_letters_images	20-11-2021 14:22	File folder	
test	26-10-2021 13:03	Microsoft Excel Co...	462 KB
train	26-10-2021 13:03	Microsoft Excel Co...	1,155 KB

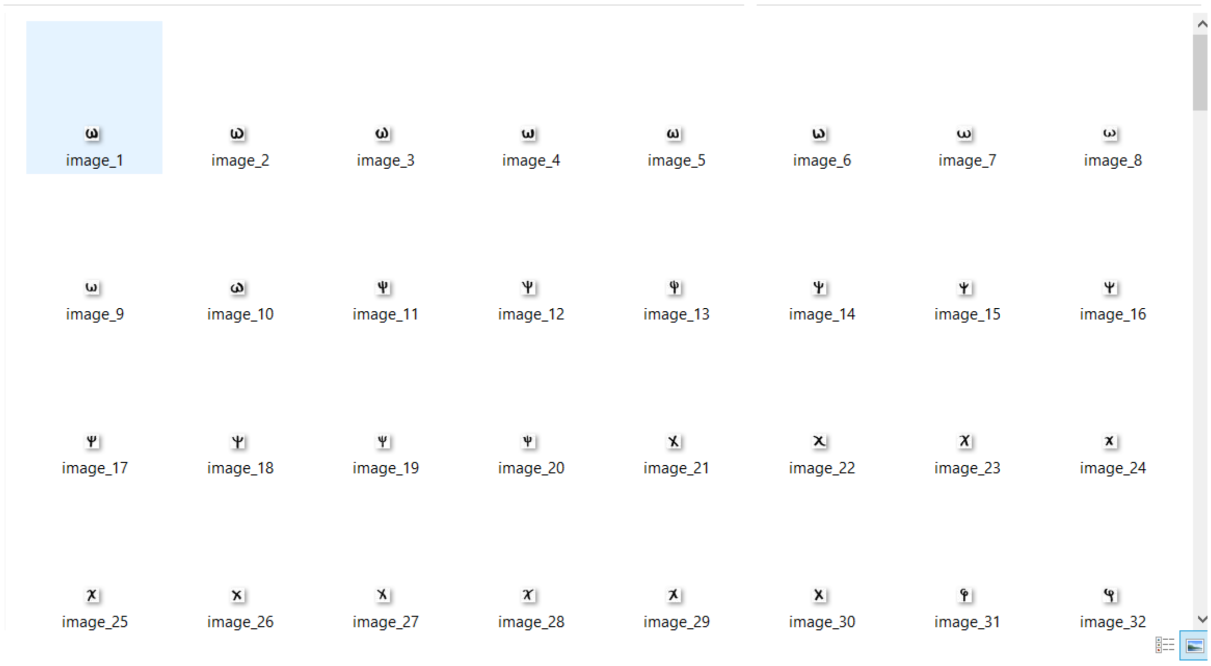
Here the train_high_resolution folder had 10 different images for each Greek character making it 240 character database (As shown in the image below)



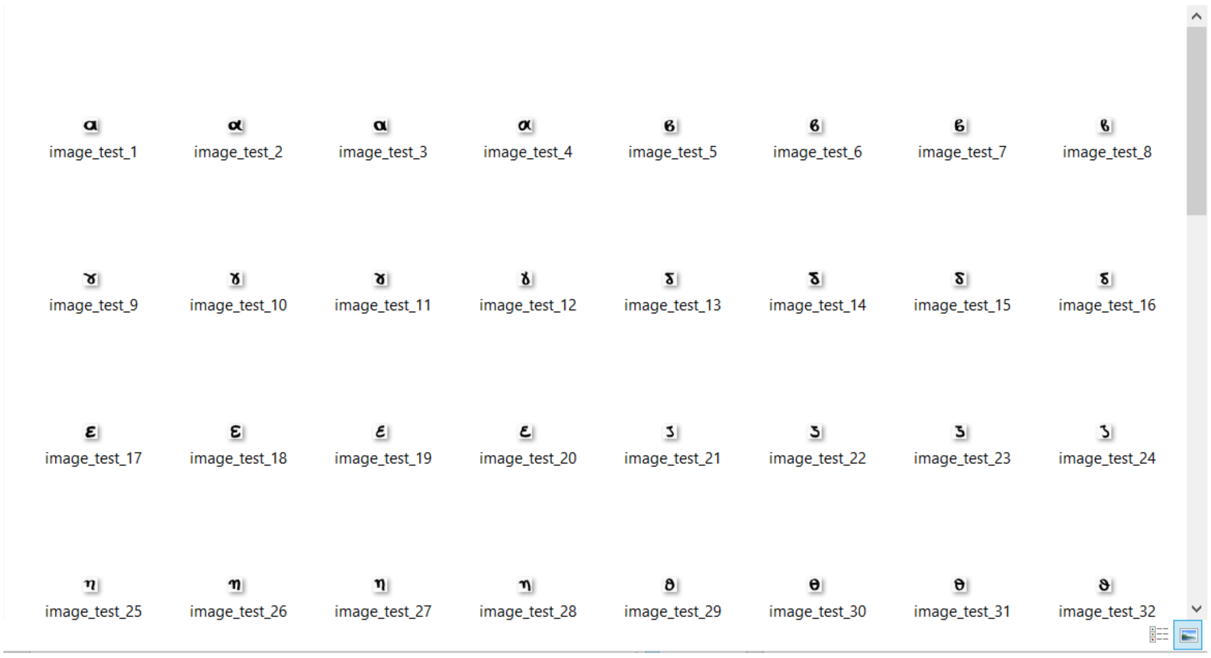
but the test_high_resolution folder had 4 different images for each Greek character making it a 96 character database which is used in this project for testing the train models. (As shown in the image below)



The same goes for train_letters_images and test_letter_images (low resolution). (As shown in the image below)

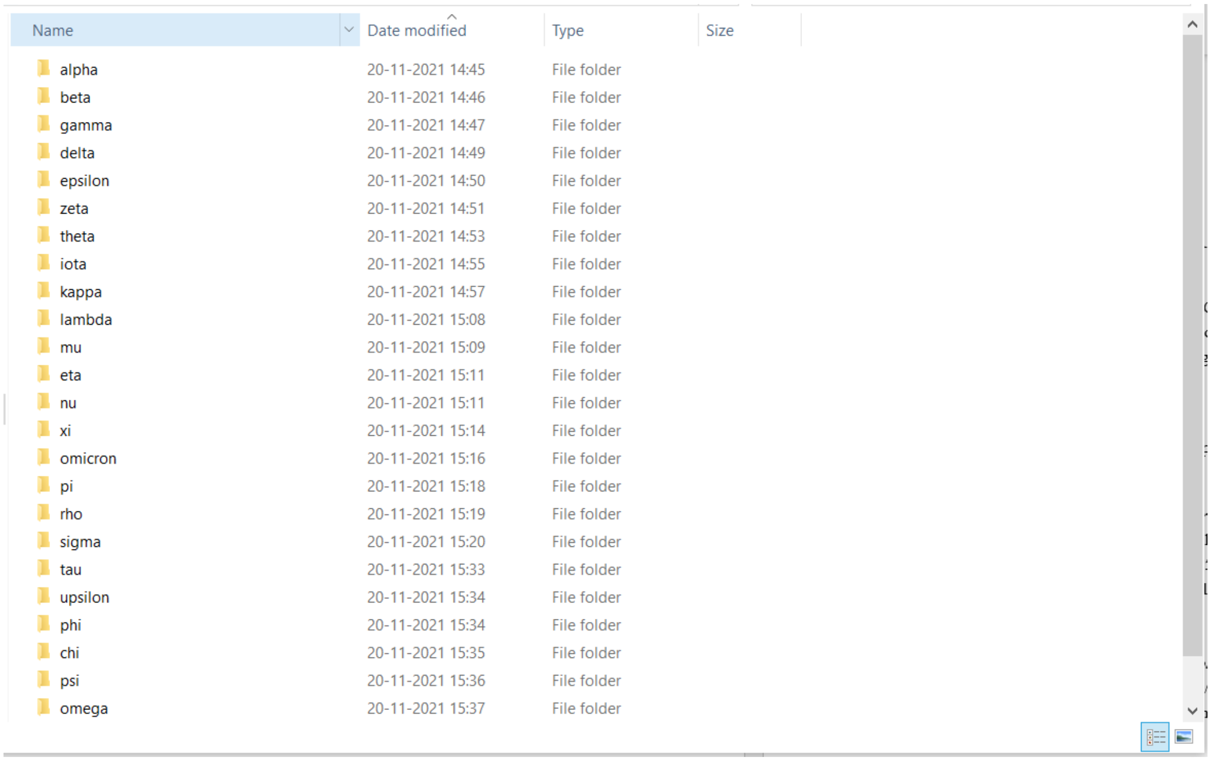


train_letter_images

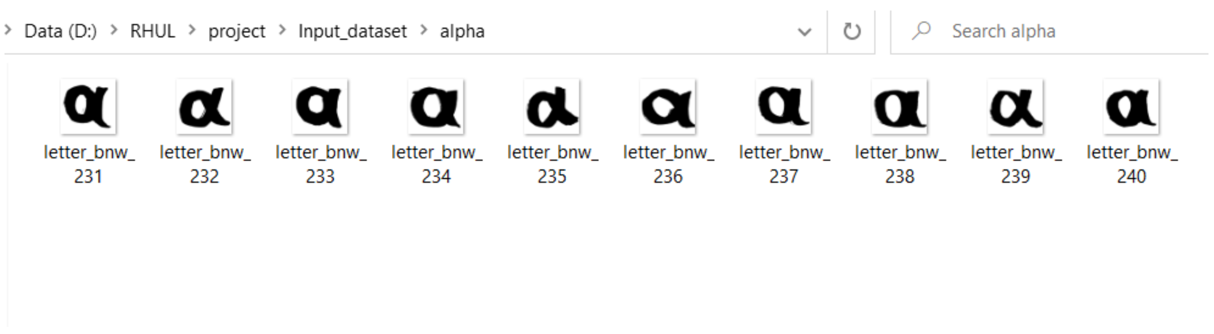

image_test_1 image_test_2 image_test_3 image_test_4 image_test_5 image_test_6 image_test_7 image_test_8
image_test_9 image_test_10 image_test_11 image_test_12 image_test_13 image_test_14 image_test_15 image_test_16
image_test_17 image_test_18 image_test_19 image_test_20 image_test_21 image_test_22 image_test_23 image_test_24
image_test_25 image_test_26 image_test_27 image_test_28 image_test_29 image_test_30 image_test_31 image_test_32

test_letter_images

The second task was to prepare the data before feeding it to the model. This was tricky and took a lot of time, I made a different folder for the High-Resolution dataset and a different Low-Resolution dataset folder in which I segregated all the 24 Greek characters into 24 different folders as shown in the image below.



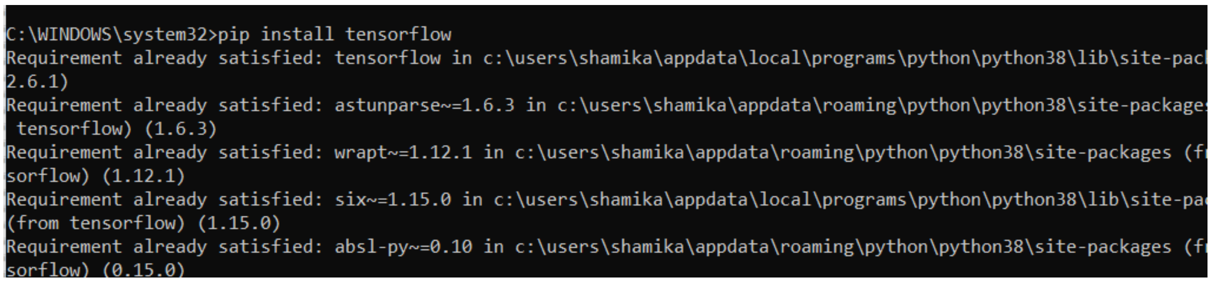
Name	Date modified	Type	Size
alpha	20-11-2021 14:45	File folder	
beta	20-11-2021 14:46	File folder	
gamma	20-11-2021 14:47	File folder	
delta	20-11-2021 14:49	File folder	
epsilon	20-11-2021 14:50	File folder	
zeta	20-11-2021 14:51	File folder	
theta	20-11-2021 14:53	File folder	
iota	20-11-2021 14:55	File folder	
kappa	20-11-2021 14:57	File folder	
lambda	20-11-2021 15:08	File folder	
mu	20-11-2021 15:09	File folder	
eta	20-11-2021 15:11	File folder	
nu	20-11-2021 15:11	File folder	
xi	20-11-2021 15:14	File folder	
omicron	20-11-2021 15:16	File folder	
pi	20-11-2021 15:18	File folder	
rho	20-11-2021 15:19	File folder	
sigma	20-11-2021 15:20	File folder	
tau	20-11-2021 15:33	File folder	
upsilon	20-11-2021 15:34	File folder	
phi	20-11-2021 15:34	File folder	
chi	20-11-2021 15:35	File folder	
psi	20-11-2021 15:36	File folder	
omega	20-11-2021 15:37	File folder	



Now that both the Input datasets (High-Resolution and Low-Resolution) are ready the training dataset is ready to be divided into a Training set and a Validation Set. The split folder module is responsible for splitting the training input data into training and validation sets, which we will talk about in the latter part of the modules section.

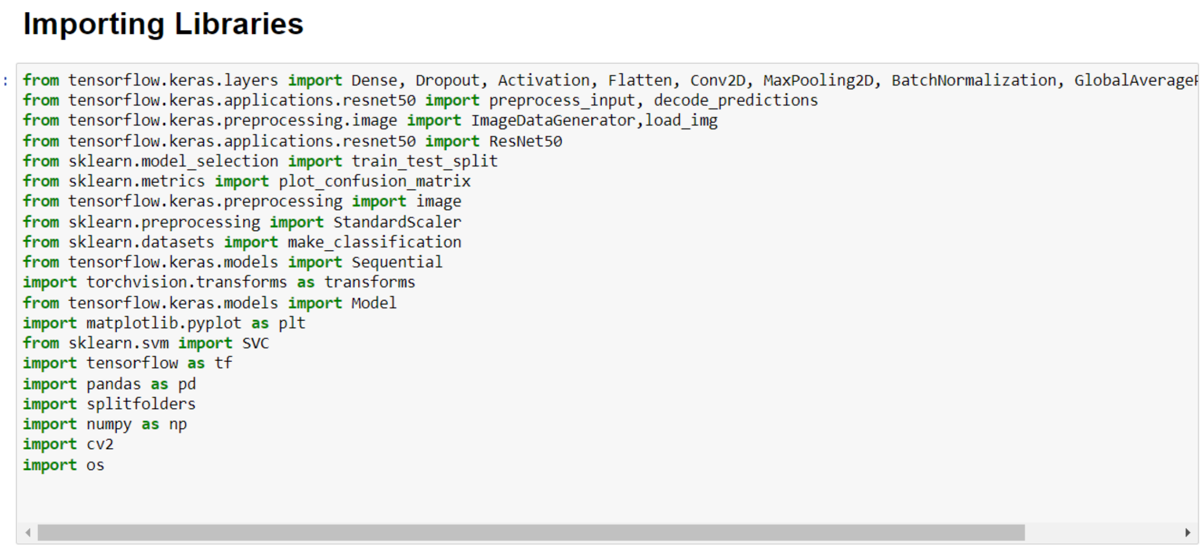
3.4 Modules

Moving forward the third step was to download and install all the modules needed for this project, which is easier to do using the **pip command.**



```
C:\WINDOWS\system32>pip install tensorflow
Requirement already satisfied: tensorflow in c:\users\shamika\appdata\local\programs\python\python38\lib\site-pac
2.6.1)
Requirement already satisfied: astunparse~=1.6.3 in c:\users\shamika\appdata\roaming\python\python38\site-package
 tensorflow) (1.6.3)
Requirement already satisfied: wrapt~=1.12.1 in c:\users\shamika\appdata\roaming\python\python38\site-packages (f
sorflow) (1.12.1)
Requirement already satisfied: six~=1.15.0 in c:\users\shamika\appdata\local\programs\python\python38\lib\site-pa
(from tensorflow) (1.15.0)
Requirement already satisfied: absl-py~=0.10 in c:\users\shamika\appdata\roaming\python\python38\site-packages (f
sorflow) (0.15.0)
```

The same goes for all the modules imported below.



Importing Libraries

```
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, BatchNormalization, GlobalAverageP
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img
from tensorflow.keras.applications.resnet50 import ResNet50
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix
from tensorflow.keras.preprocessing import image
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification
from tensorflow.keras.models import Sequential
import torchvision.transforms as transforms
from tensorflow.keras.models import Model
import matplotlib.pyplot as plt
from sklearn.svm import SVC
import tensorflow as tf
import pandas as pd
import splitfolders
import numpy as np
import cv2
import os
```

tensorflow.keras.layers are imported for using the Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D as the layers for the Convolutional Neural Network.

prediction_input and decode_predictions are used for predicting the labels of the test image.

ImageDataGenerator and load_img are used for loading the images from the datasets.

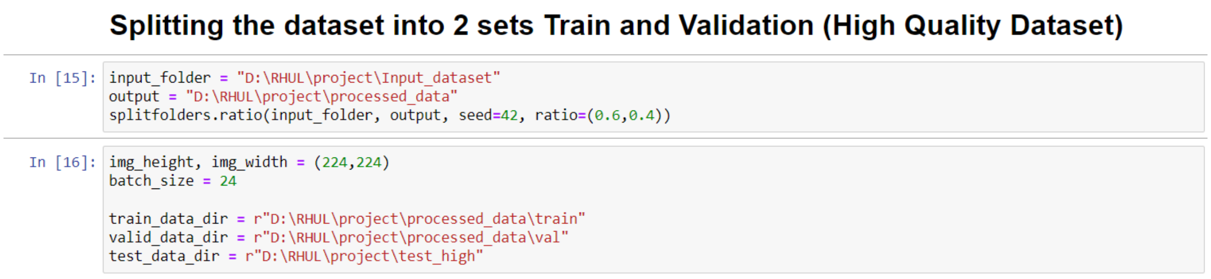
The Model module helps us to load the pre-trained models needed for this project that are the ResNet50 and CNN.

make_classification, train_test_split, plot_confusion_matrix are imported to plot a confusion matrix for the x_train, x_test, y_train and y_test making the label prediction visualization easy to understand.

cv2 module is imported for loading the train and test image from a specific folder. Which is later it is used for visualizing purposes in this project.

The os module is imported as it is responsible to provide the function for the program to interact with the operating system. This module comes under the python system's utility module making the user to access the operating system with more ease.

numpy is the module used in python for all mathematical functionalities.



Splitting the dataset into 2 sets Train and Validation (High Quality Dataset)

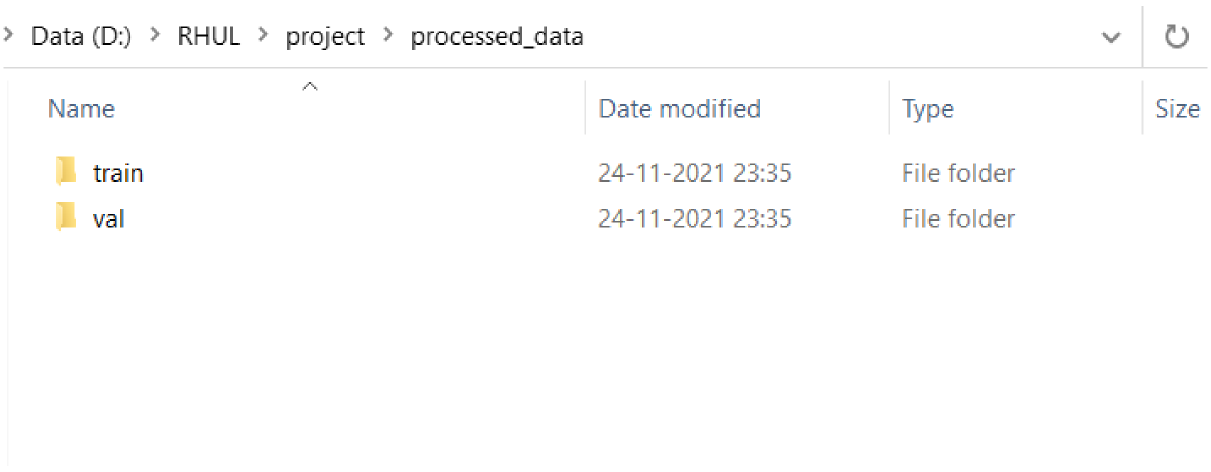
```
In [15]: input_folder = "D:\RHUL\project\Input_dataset"
         output = "D:\RHUL\project\processed_data"
         splitfolders.ratio(input_folder, output, seed=42, ratio=(0.6,0.4))
```

```
In [16]: img_height, img_width = (224,224)
         batch_size = 24

         train_data_dir = r"D:\RHUL\project\processed_data\train"
         valid_data_dir = r"D:\RHUL\project\processed_data\val"
         test_data_dir = r"D:\RHUL\project\test_high"
```

The splitfolders module is used to split the High resolution and low resolution Greek characters data set which was made earlier into 2 sets that are the train set and the validation set. In the above image, we can see the Input_dataset is initialized as the input folder where we provide it with its path then we provide the path and the folder's name where the output should be gone after the splitting process. ratio() function of the splitfolders module the image height and width is initialized as (224,224) and as we know there are 24 greek characters the batch size is 24.

```
splitfolders.ratio(input_folder, output, seed=42, ratio=(0.6,0.4))
```

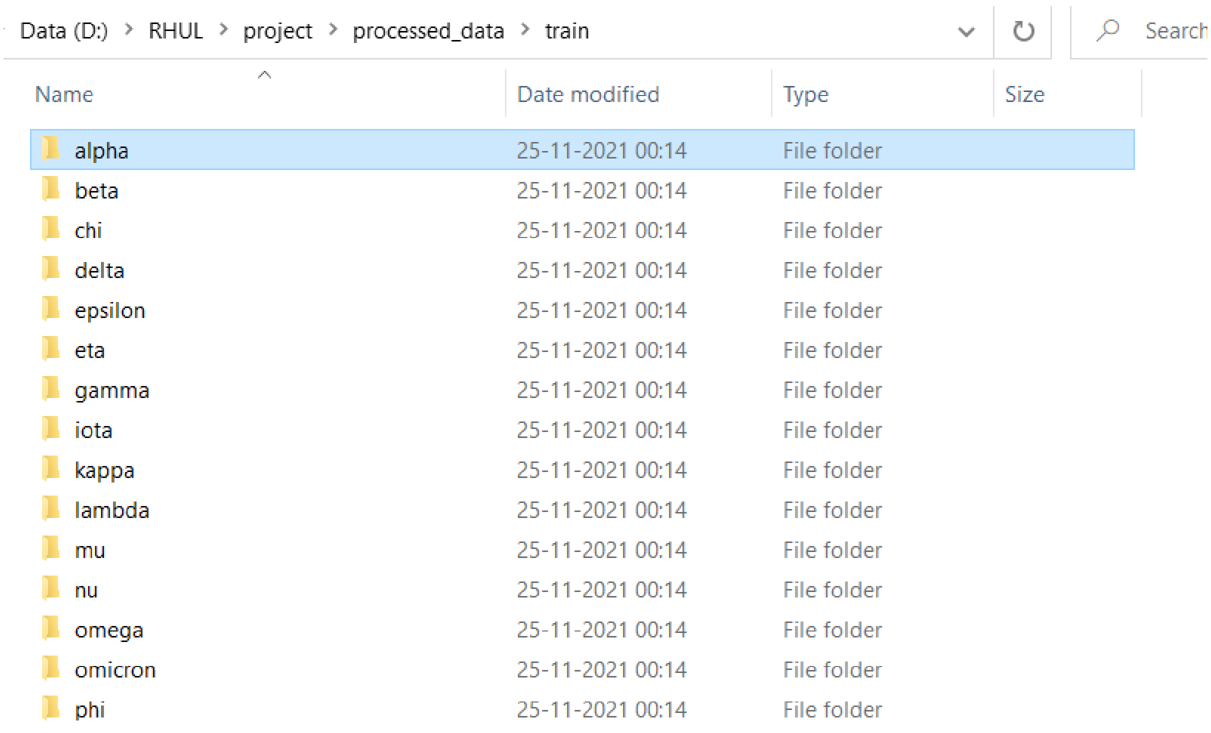
This line signifies that the dataset in the input folder should be splitted into a ratio of 60% and 40% and this data should be processed into the output that is the processed data folder.



Data (D:) > RHUL > project > processed_data

Name	Date modified	Type	Size
train	24-11-2021 23:35	File folder	
val	24-11-2021 23:35	File folder	

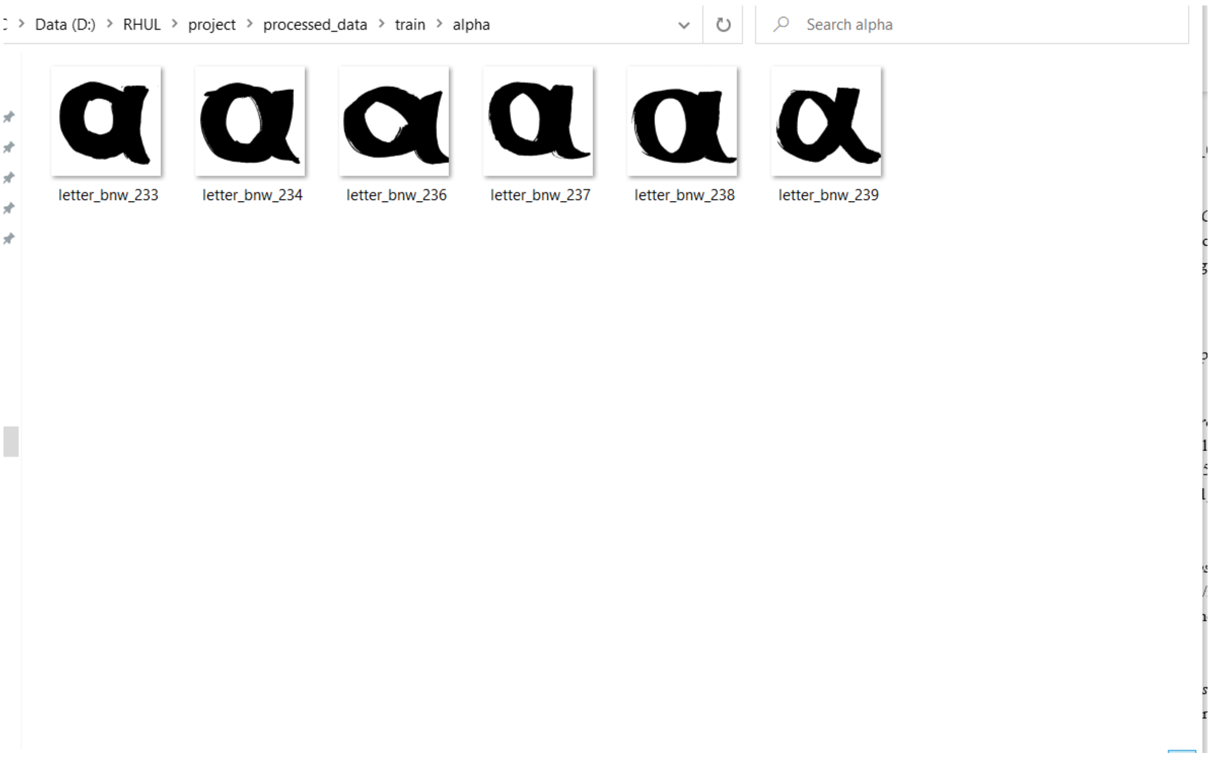
In the above image, it is seen that train and val folders are made in the processes_data folder.



Data (D:) › RHUL › project › processed_data › train

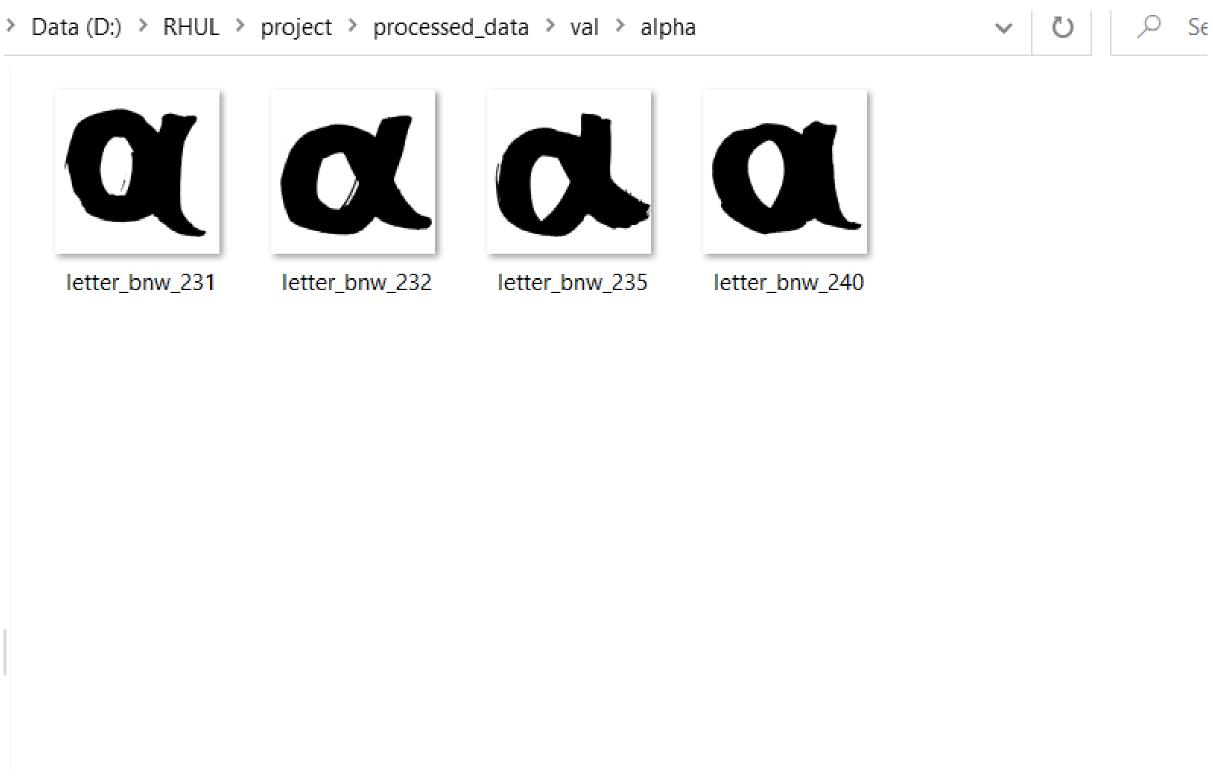
Name	Date modified	Type	Size
alpha	25-11-2021 00:14	File folder	
beta	25-11-2021 00:14	File folder	
chi	25-11-2021 00:14	File folder	
delta	25-11-2021 00:14	File folder	
epsilon	25-11-2021 00:14	File folder	
eta	25-11-2021 00:14	File folder	
gamma	25-11-2021 00:14	File folder	
iota	25-11-2021 00:14	File folder	
kappa	25-11-2021 00:14	File folder	
lambda	25-11-2021 00:14	File folder	
mu	25-11-2021 00:14	File folder	
nu	25-11-2021 00:14	File folder	
omega	25-11-2021 00:14	File folder	
omicron	25-11-2021 00:14	File folder	
phi	25-11-2021 00:14	File folder	

The train and val folders have the same 24 greek character folders the only difference is that the percentage of splitting is different.



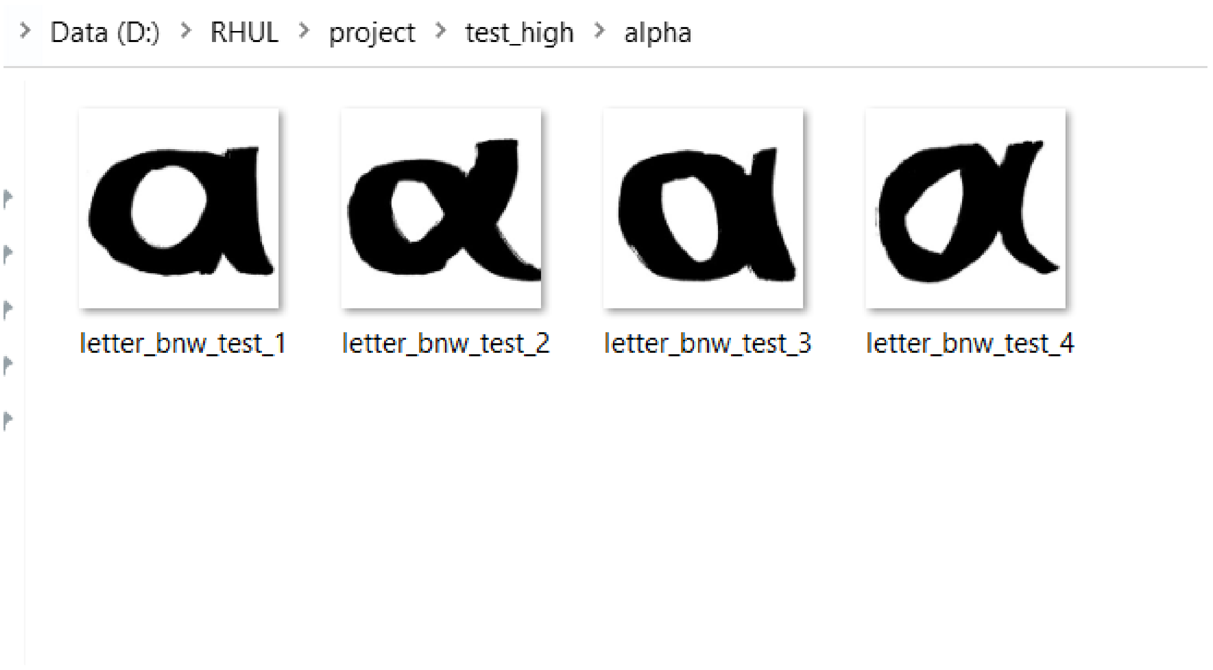
Data (D:) › RHUL › project › processed_data › train › alpha

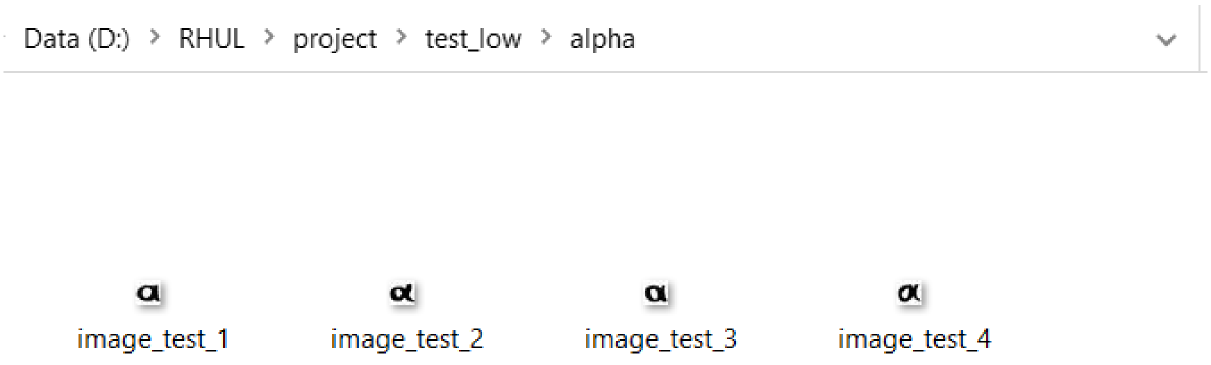
letter_bnw_233 letter_bnw_234 letter_bnw_236 letter_bnw_237 letter_bnw_238 letter_bnw_239



Both the images show that in the training dataset we have 6 images and the validation set has 4 images, this proves over code is working fine and the input_dataset is split into a ratio of 0.6 and 0.4.

Going ahead in the datasets the test set is kept completely different to avoid leaking of data and overfitting the results.

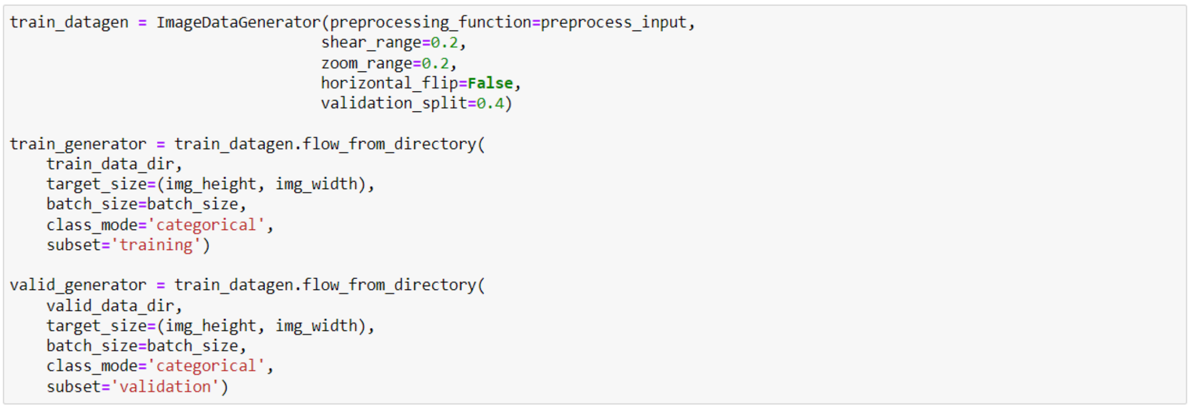




test_high and test_low contain 4 images each in 24 folders named alpha, beta, gamma, etc. till omega which are further used to evaluate the models. This concludes the modules subsection where dataset splitting is also explained.

3.5 Image Augmentation

ImageDataGenerator is used to augment the images meaning altering or adding the data to model it for training purposes. The images that are augmented using the ImageDataGenerator are given as a feed to the model which is going to be trained.

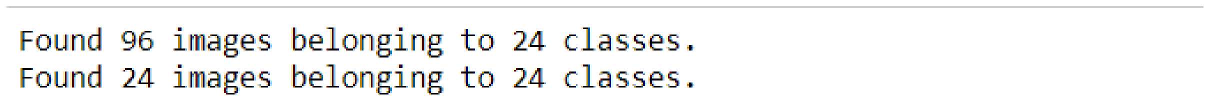


```
train_datagen = ImageDataGenerator(preprocessing_function=preprocess_input,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=False,
                                   validation_split=0.4)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training')

valid_generator = train_datagen.flow_from_directory(
    valid_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation')
```

In the above image train_datagen is the variable used to assign all the changes made with the help of the ImageDataGenerator.

Then the train_generator is given images of the inputs from train_data_dir which we have seen earlier, the directory is accessed with the help of train_datagen.flow_from_directory. train_data_dir and valid_data_dir provide the path for where the training image dataset and validation image dataset, target size specifies image height and image width, which is specified earlier, the batch size is also specified earlier which is 24 for 24 Greek characters. Class mode is set to categorical as the Greek dataset has 24 categories.

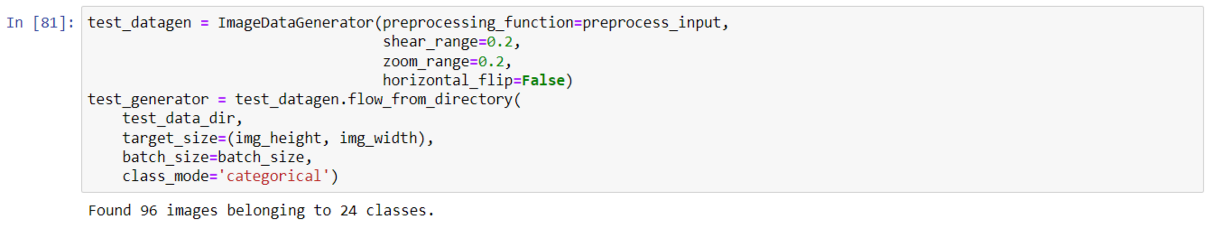
This code gives an output stated in the image below.



```
Found 96 images belonging to 24 classes.
Found 24 images belonging to 24 classes.
```

This gives proof that our 60 and 40 percent split of training and validation dataset is correctly executed as there are 240 total images in the input dataset which is further divided into 96 images belonging to 24 classes for the training dataset and 24 images belonging to 24 classes for the validation dataset.

For the test set, the code changes as shown in the image below

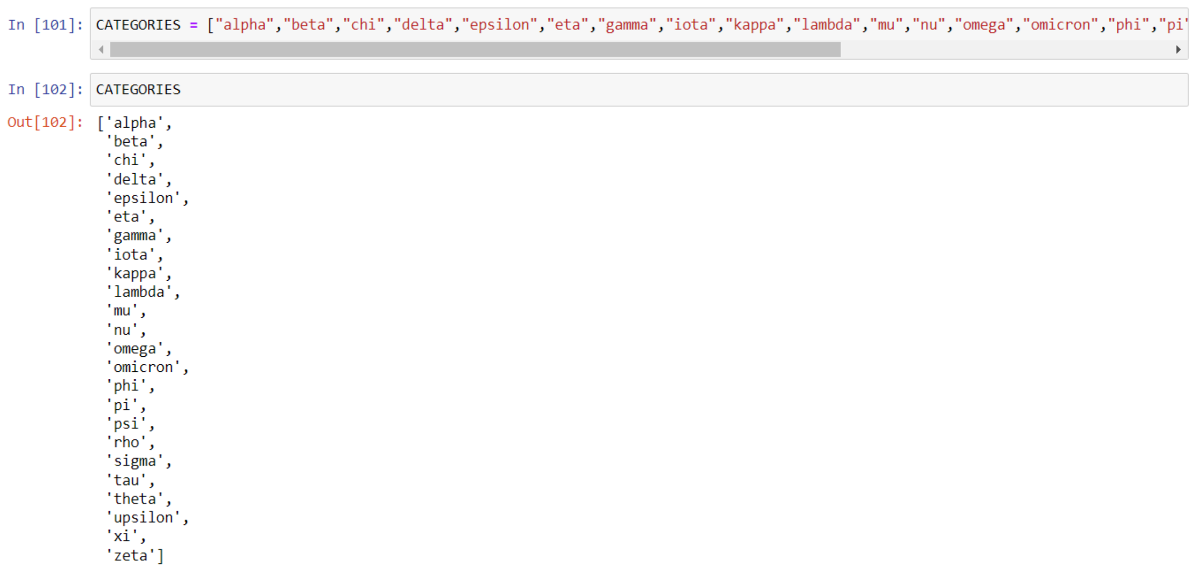


```
In [81]: test_datagen = ImageDataGenerator(preprocessing_function=preprocess_input,
                                           shear_range=0.2,
                                           zoom_range=0.2,
                                           horizontal_flip=False)
         test_generator = test_datagen.flow_from_directory(
             test_data_dir,
             target_size=(img_height, img_width),
             batch_size=batch_size,
             class_mode='categorical')

         Found 96 images belonging to 24 classes.
```

Here validation-split is not specified as we need to use the whole test dataset which contains 4 images of each category which is a total of 24 characters.

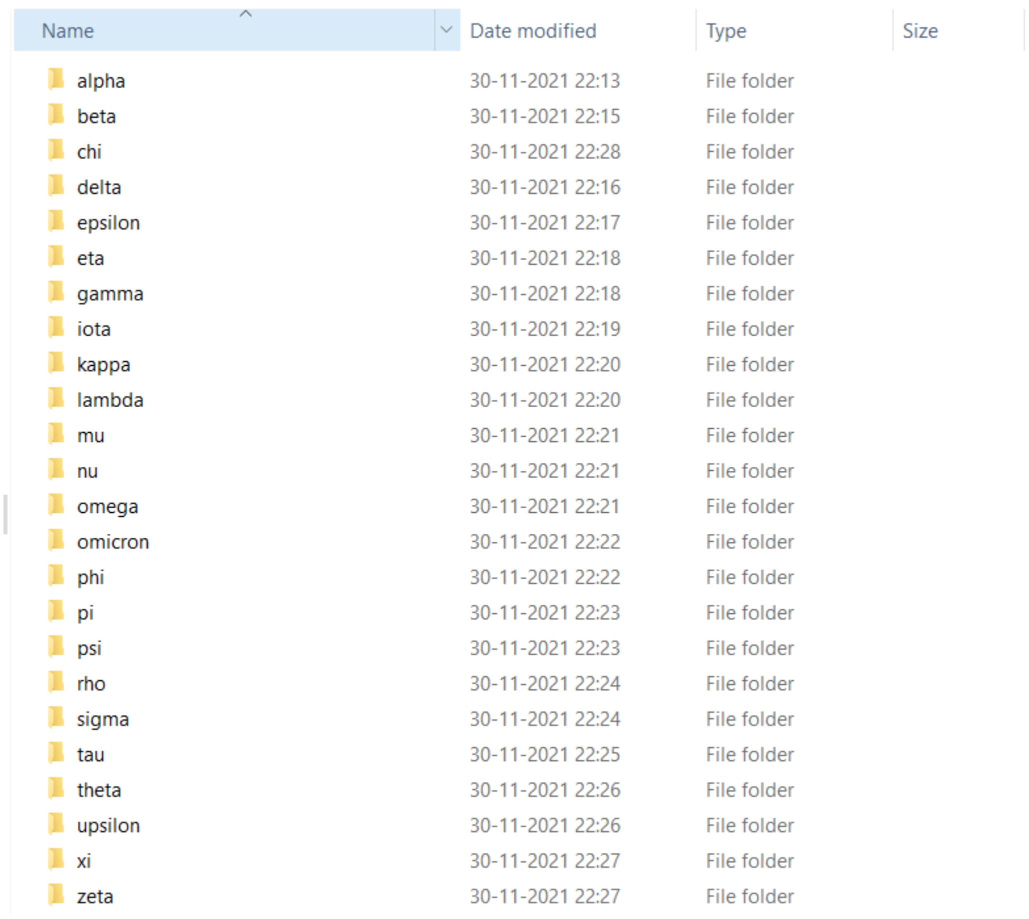
3.6 Initializing the categories

Initializing the categories play an important role while comparing the True labels to the Predicted labels.



```
In [101]: CATEGORIES = ["alpha","beta","chi","delta","epsilon","eta","gamma","iota","kappa","lambda","mu","nu","omega","omicron","phi","pi'

In [102]: CATEGORIES

Out[102]: ['alpha',
           'beta',
           'chi',
           'delta',
           'epsilon',
           'eta',
           'gamma',
           'iota',
           'kappa',
           'lambda',
           'mu',
           'nu',
           'omega',
           'omicron',
           'phi',
           'pi',
           'psi',
           'rho',
           'sigma',
           'tau',
           'theta',
           'upsilon',
           'xi',
           'zeta']
```

As shown in the above figure the categories are assigned according to the numbering of the folders shown in the image below.



Name	Date modified	Type	Size
alpha	30-11-2021 22:13	File folder	
beta	30-11-2021 22:15	File folder	
chi	30-11-2021 22:28	File folder	
delta	30-11-2021 22:16	File folder	
epsilon	30-11-2021 22:17	File folder	
eta	30-11-2021 22:18	File folder	
gamma	30-11-2021 22:18	File folder	
iota	30-11-2021 22:19	File folder	
kappa	30-11-2021 22:20	File folder	
lambda	30-11-2021 22:20	File folder	
mu	30-11-2021 22:21	File folder	
nu	30-11-2021 22:21	File folder	
omega	30-11-2021 22:21	File folder	
omicron	30-11-2021 22:22	File folder	
phi	30-11-2021 22:22	File folder	
pi	30-11-2021 22:23	File folder	
psi	30-11-2021 22:23	File folder	
rho	30-11-2021 22:24	File folder	
sigma	30-11-2021 22:24	File folder	
tau	30-11-2021 22:25	File folder	
theta	30-11-2021 22:26	File folder	
upsilon	30-11-2021 22:26	File folder	
xi	30-11-2021 22:27	File folder	
zeta	30-11-2021 22:27	File folder	

This means 0 is assigned to alpha,

1 is assigned to beta,

2 is assigned to chi,

3 is assigned to delta,

4 is assigned to epsilon,

5 is assigned to eta,

6 is assigned to gamma,

7 is assigned to iota,

8 is assigned to kappa,

9 is assigned to lambda,

10 is assigned to mu,

11 is assigned to nu,

12 is assigned to omega,

13 is assigned to omicron,

14 is assigned to phi,

15 is assigned to pi,

16 is assigned to psi,

17 is assigned to rho,

18 is assigned to sigma,

19 is assigned to tau,

20 is assigned to theta,

21 is assigned to upsilon,

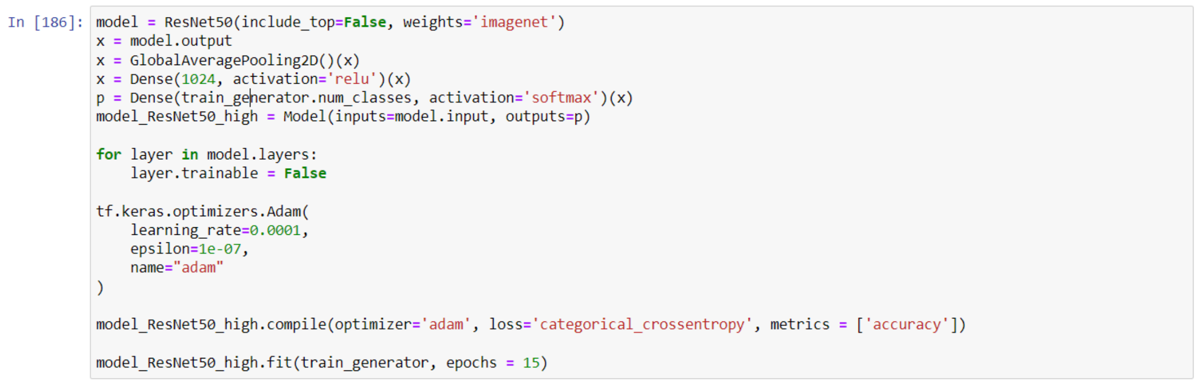
22 is assigned to xi,

23 is assigned to zeta

All are assigned a number alphabetically.

3.7 Training

3.7.1 ResNet50 model with High resolution Greek Character training dataset.



```
In [186]: model = ResNet50(include_top=False, weights='imagenet')
          x = model.output
          x = GlobalAveragePooling2D()(x)
          x = Dense(1024, activation='relu')(x)
          p = Dense(train_generator.num_classes, activation='softmax')(x)
          model_ResNet50_high = Model(inputs=model.input, outputs=p)

          for layer in model.layers:
              layer.trainable = False

          tf.keras.optimizers.Adam(
              learning_rate=0.0001,
              epsilon=1e-07,
              name="adam"
          )

          model_ResNet50_high.compile(optimizer='adam', loss='categorical_crossentropy', metrics = ['accuracy'])

          model_ResNet50_high.fit(train_generator, epochs = 15)
```

ResNet50 is an inbuilt model in TensorFlow which is used in the above image, ImageNet is specified as weights for this model, and ResNet50 works pretty well with the image dataset it is also easy to build.

The first activation function given is ReLu which takes only 1 and any positive number greater than 1, and if it is given 0 or any negative number as input it switches of the node.

The last activation function given is SoftMax which is explained in detail in the Activation Function subsection.

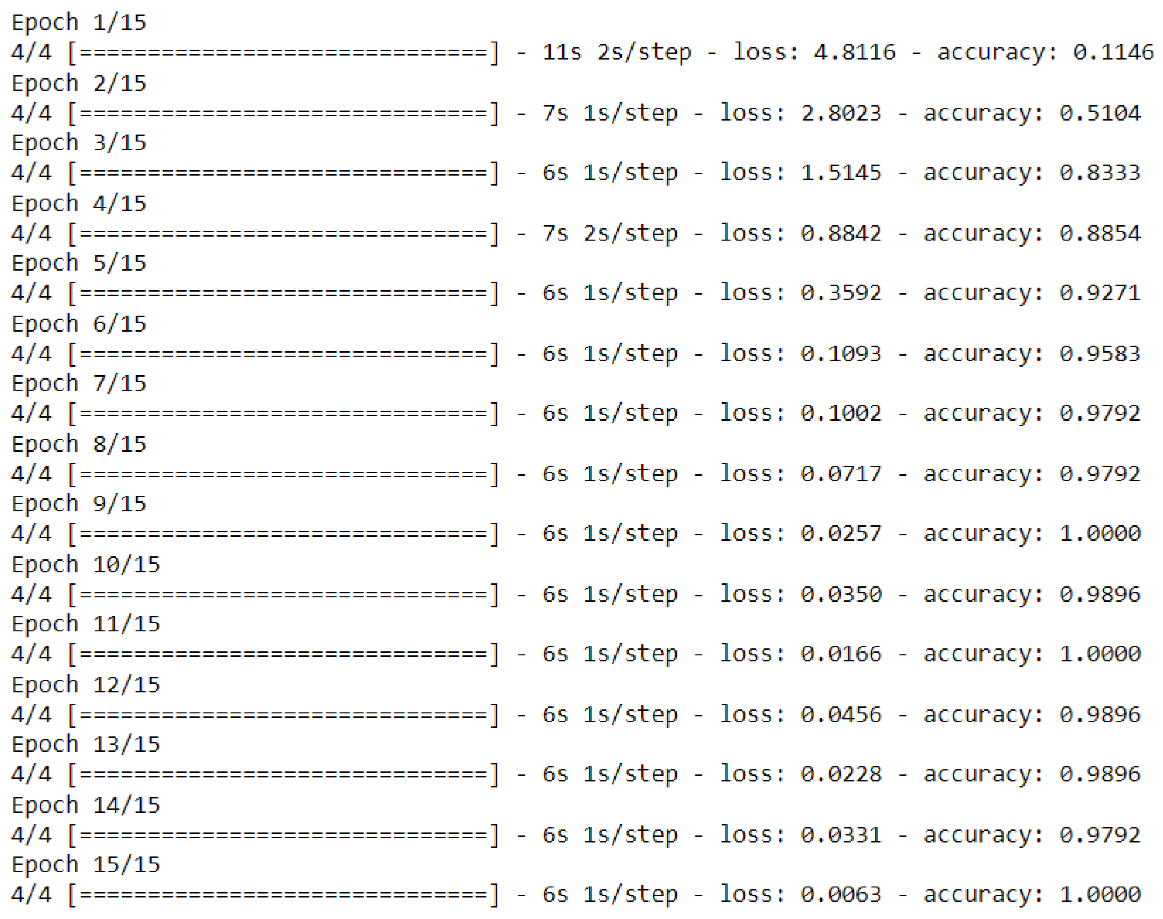
train_generator.num_classes specifies the number of total classes in the Handwritten Greek Character dataset, which is 24.

layer.trainable is initialized False to freeze the layer making it shift from trainable weights to non-trainable weights this helps a lot to get good accuracy.

The optimizers in deep learning are used for fine tuning the model to get good results on the test set, here the optimizer used is *Adam* which is the most commonly used to optimize. The learning rate is initialized as **0.0001** the smaller the learning rate is better for good accuracy. Epsilon is initialized to 1e-07, epsilon should be a small value as it is responsible to tackle the error raised due to dividing by zero.

model.compile function is used to compile the final specification that are the optimizer used, the loss that can be specified as **binary cross entropy or categorical cross entropy (which is used in this project).** Categorical cross entropy is used when we are dealing with Multi class image dataset.

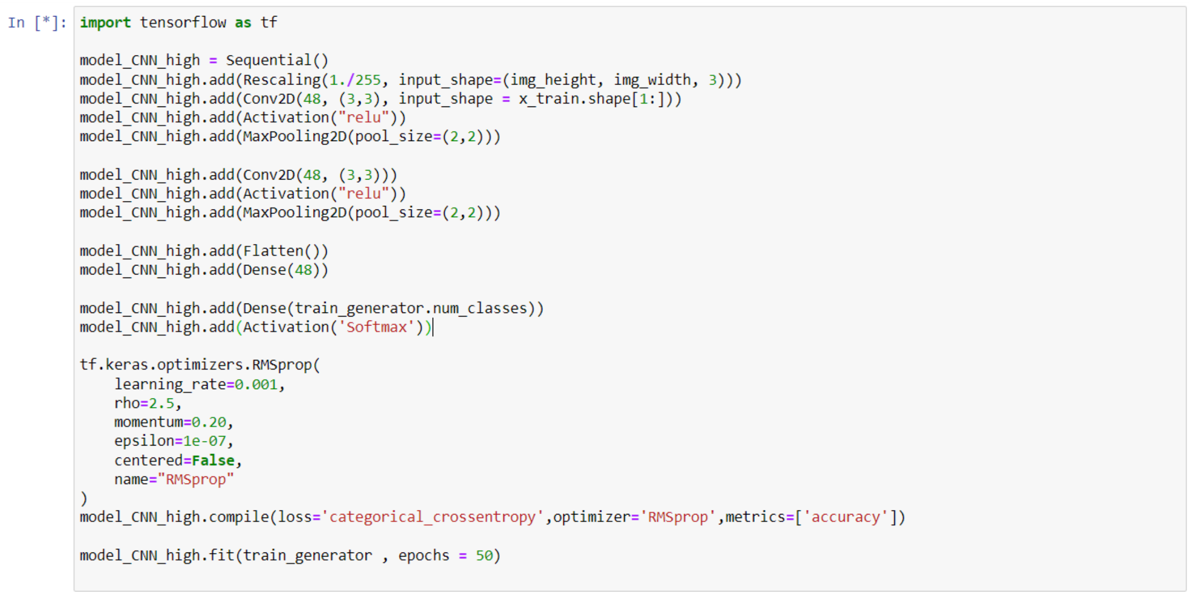
Then in the end the model created is fitted with train_generator as input and with 15 epochs with the help of model.fit() function.



```
Epoch 1/15
4/4 [==============================] - 11s 2s/step - loss: 4.8116 - accuracy: 0.1146
Epoch 2/15
4/4 [==============================] - 7s 1s/step - loss: 2.8023 - accuracy: 0.5104
Epoch 3/15
4/4 [==============================] - 6s 1s/step - loss: 1.5145 - accuracy: 0.8333
Epoch 4/15
4/4 [==============================] - 7s 2s/step - loss: 0.8842 - accuracy: 0.8854
Epoch 5/15
4/4 [==============================] - 6s 1s/step - loss: 0.3592 - accuracy: 0.9271
Epoch 6/15
4/4 [==============================] - 6s 1s/step - loss: 0.1093 - accuracy: 0.9583
Epoch 7/15
4/4 [==============================] - 6s 1s/step - loss: 0.1002 - accuracy: 0.9792
Epoch 8/15
4/4 [==============================] - 6s 1s/step - loss: 0.0717 - accuracy: 0.9792
Epoch 9/15
4/4 [==============================] - 6s 1s/step - loss: 0.0257 - accuracy: 1.0000
Epoch 10/15
4/4 [==============================] - 6s 1s/step - loss: 0.0350 - accuracy: 0.9896
Epoch 11/15
4/4 [==============================] - 6s 1s/step - loss: 0.0166 - accuracy: 1.0000
Epoch 12/15
4/4 [==============================] - 6s 1s/step - loss: 0.0456 - accuracy: 0.9896
Epoch 13/15
4/4 [==============================] - 6s 1s/step - loss: 0.0228 - accuracy: 0.9896
Epoch 14/15
4/4 [==============================] - 6s 1s/step - loss: 0.0331 - accuracy: 0.9792
Epoch 15/15
4/4 [==============================] - 6s 1s/step - loss: 0.0063 - accuracy: 1.0000
```

The above image shows that after 15 Epochs and due to good accuracy, our model has fitted well with the high resolution Greek character dataset.

3.7.2 CNN model with High resolution Greek Character training dataset.



```
In [*]: import tensorflow as tf

        model_CNN_high = Sequential()
        model_CNN_high.add(Rescaling(1./255, input_shape=(img_height, img_width, 3)))
        model_CNN_high.add(Conv2D(48, (3,3), input_shape = x_train.shape[1:]))
        model_CNN_high.add(Activation("relu"))
        model_CNN_high.add(MaxPooling2D(pool_size=(2,2)))

        model_CNN_high.add(Conv2D(48, (3,3)))
        model_CNN_high.add(Activation("relu"))
        model_CNN_high.add(MaxPooling2D(pool_size=(2,2)))

        model_CNN_high.add(Flatten())
        model_CNN_high.add(Dense(48))

        model_CNN_high.add(Dense(train_generator.num_classes))
        model_CNN_high.add(Activation('Softmax'))

        tf.keras.optimizers.RMSprop(
            learning_rate=0.001,
            rho=2.5,
            momentum=0.20,
            epsilon=1e-07,
            centered=False,
            name="RMSprop"
        )
        model_CNN_high.compile(loss='categorical_crossentropy',optimizer='RMSprop',metrics=['accuracy'])

        model_CNN_high.fit(train_generator , epochs = 50)
```

CNN is an inbuilt model in TensorFlow which has Conv2D, Activation, Maxpooling2D as the layers in tensorflow.keras.layers which are used in the above image.

Rescaling is used to process the image input, normalize it by dividing it with 255.0, and give the input shape as specified while pre-processing the image dataset with a dimension of 3 dim.

The first activation function given is ReLu which takes only 1 and any positive number greater than 1, and if it is given 0 or any negative number as input it switches of the node.

The Max pooling layer is responsible for the Suppression of the noise produced during the training.

The last activation function given is SoftMax which is explained in detail in the Activation Function subsection.

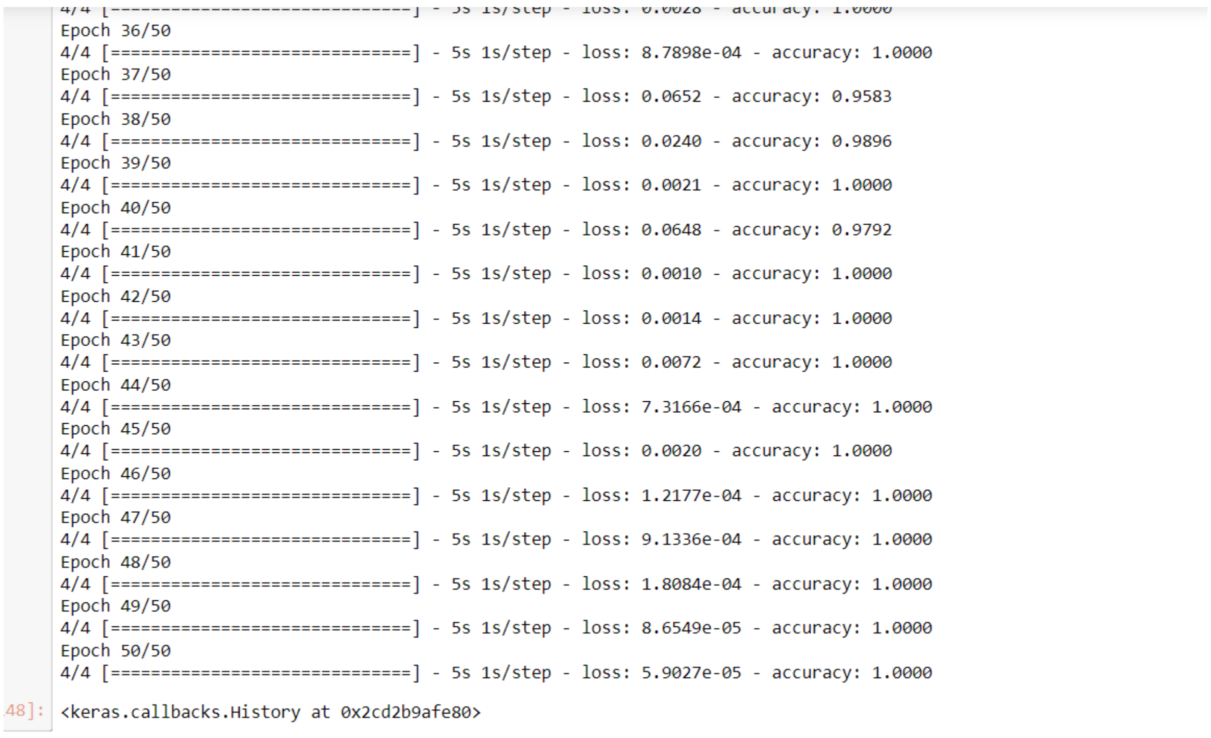
train_generator.num_classes specifies the number of total classes in the Handwritten Greek Character dataset, which is 24.

For CNN initializing layer.trainable to False doesn't work as the accuracy gets stuck to repeating the same accuracy again and again.

The optimizers in deep learning are used for fine tuning the model to get good results on the test set, here the optimizer used is *RmsProp* which is also the most commonly used to optimize. Rho is specified as **2.5** which is responsible for calculating the potential weighted average over the square of the gradients. The momentum is used to overcome the problem of noisy gradients which is specified to **0.20**.The learning rate is initialized as **0.001** the smaller the learning rate is better for good accuracy. Epsilon is initialized to **1e-07**, epsilon should be a small value as it is responsible to tackle the error raised due to dividing by zero.

model.compile function is used to compile the final specification that are the optimizer used, the loss that can be specified as **binary cross entropy or categorical cross entropy (which is used in this project).** Categorical cross entropy is used when we are dealing with Multi class image dataset.

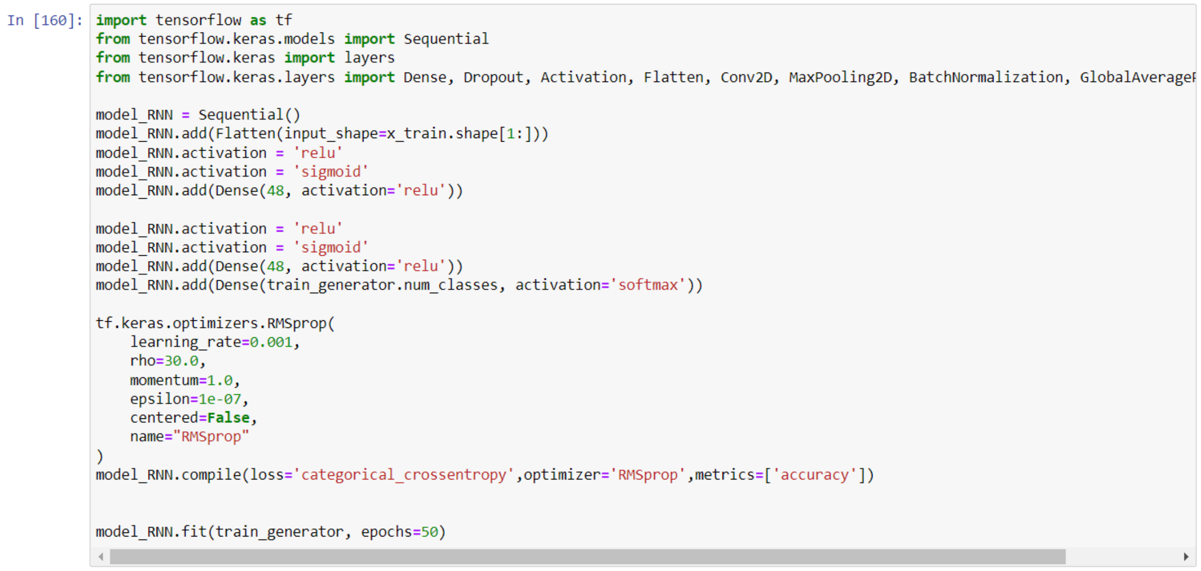
Then in the end the model created is fitted with train_generator as input and with 50 epochs with the help of model.fit() function.



```
4/4 [==============================] - 5s 1s/step - loss: 0.0028 - accuracy: 1.0000
Epoch 36/50
4/4 [==============================] - 5s 1s/step - loss: 8.7898e-04 - accuracy: 1.0000
Epoch 37/50
4/4 [==============================] - 5s 1s/step - loss: 0.0652 - accuracy: 0.9583
Epoch 38/50
4/4 [==============================] - 5s 1s/step - loss: 0.0240 - accuracy: 0.9896
Epoch 39/50
4/4 [==============================] - 5s 1s/step - loss: 0.0021 - accuracy: 1.0000
Epoch 40/50
4/4 [==============================] - 5s 1s/step - loss: 0.0648 - accuracy: 0.9792
Epoch 41/50
4/4 [==============================] - 5s 1s/step - loss: 0.0010 - accuracy: 1.0000
Epoch 42/50
4/4 [==============================] - 5s 1s/step - loss: 0.0014 - accuracy: 1.0000
Epoch 43/50
4/4 [==============================] - 5s 1s/step - loss: 0.0072 - accuracy: 1.0000
Epoch 44/50
4/4 [==============================] - 5s 1s/step - loss: 7.3166e-04 - accuracy: 1.0000
Epoch 45/50
4/4 [==============================] - 5s 1s/step - loss: 0.0020 - accuracy: 1.0000
Epoch 46/50
4/4 [==============================] - 5s 1s/step - loss: 1.2177e-04 - accuracy: 1.0000
Epoch 47/50
4/4 [==============================] - 5s 1s/step - loss: 9.1336e-04 - accuracy: 1.0000
Epoch 48/50
4/4 [==============================] - 5s 1s/step - loss: 1.8084e-04 - accuracy: 1.0000
Epoch 49/50
4/4 [==============================] - 5s 1s/step - loss: 8.6549e-05 - accuracy: 1.0000
Epoch 50/50
4/4 [==============================] - 5s 1s/step - loss: 5.9027e-05 - accuracy: 1.0000
48]: <keras.callbacks.History at 0x2cd2b9afe80>
```

The above image shows that after 50 Epochs and due to good accuracy, our model has fitted well with the high resolution Greek character dataset.

3.7.3 RNN model with High resolution Greek Character training dataset.



```
In [160]: import tensorflow as tf
          from tensorflow.keras.models import Sequential
          from tensorflow.keras import layers
          from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, BatchNormalization, GlobalAverageP

          model_RNN = Sequential()
          model_RNN.add(Flatten(input_shape=x_train.shape[1:]))
          model_RNN.activation = 'relu'
          model_RNN.activation = 'sigmoid'
          model_RNN.add(Dense(48, activation='relu'))

          model_RNN.activation = 'relu'
          model_RNN.activation = 'sigmoid'
          model_RNN.add(Dense(48, activation='relu'))
          model_RNN.add(Dense(train_generator.num_classes, activation='softmax'))

          tf.keras.optimizers.RMSprop(
              learning_rate=0.001,
              rho=30.0,
              momentum=1.0,
              epsilon=1e-07,
              centered=False,
              name="RMSprop"
          )
          model_RNN.compile(loss='categorical_crossentropy',optimizer='RMSprop',metrics=['accuracy'])


          model_RNN.fit(train_generator, epochs=50)
```

Here RNN is built from scratch and TensorFlow has Dense and Flatten as the layers in tensorflow.keras.layers which are used in the above image.

The first activation function given is ReLu which takes only 1 and any positive number greater than 1, and if it is given 0 or any negative number as input it switches of the node.

Flatten is used to manipulate the input shape.

The Sigmoid function is also an activation function used to convert the input into probability which can also be stated as mapping the values into very small numbers that are between 0 to 1 or -1 to 1

The last activation function given is SoftMax which is explained in detail in the Activation Function subsection.

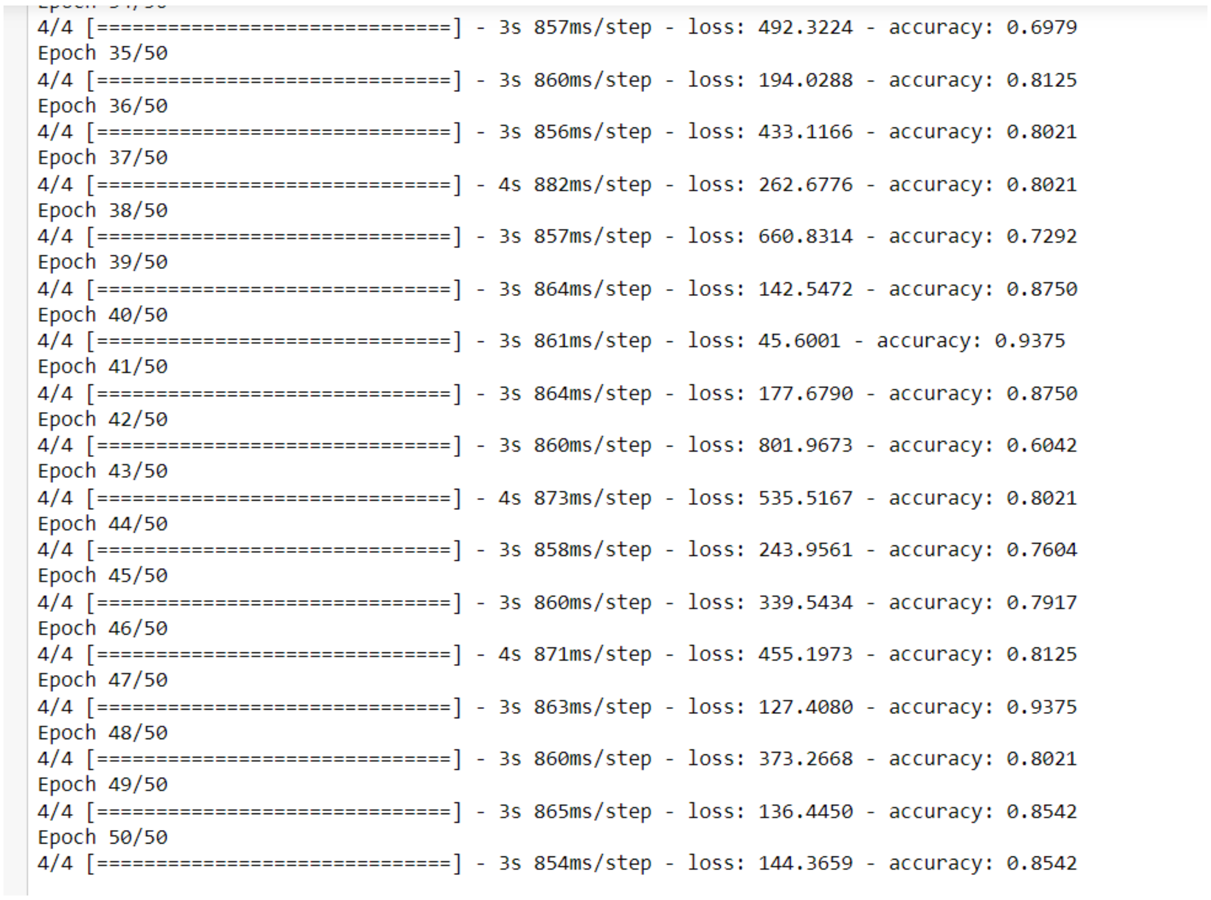
train_generator.num_classes specifies the number of total classes in the Handwritten Greek Character dataset, which is 24.

For RNN initializing layer.trainable to False doesn't work as the accuracy gets stuck to repeating the same accuracy again and again.

The optimizers in deep learning are used for fine tuning the model to get good results on the test set, here the optimizer used is *RmsProp* which is also the most commonly used to optimize. Rho is specified as **30.0** which is responsible for calculating the potential weighted average over the square of the gradients. The momentum is used to overcome the problem of noisy gradients which is specified to **1.0**. The learning rate is initialized as **0.001** the smaller the learning rate is better for good accuracy. Epsilon is initialized to **1e-07**, epsilon should be a small value as it is responsible to tackle the error raised due to dividing by zero.

model.compile function is used to compile the final specification that are the optimizer used, the loss that can be specified as **binary cross entropy or categorical cross entropy (which is used in this project).** Categorical cross entropy is used when we are dealing with Multi class image dataset.

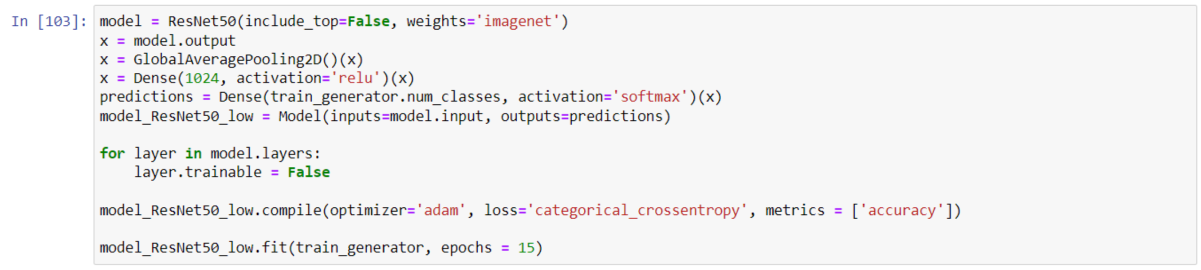
Then in the end the model created is fitted with train_generator as input and with 50 epochs with the help of model.fit() function.



```
4/4 [==============================] - 3s 857ms/step - loss: 492.3224 - accuracy: 0.6979
Epoch 35/50
4/4 [==============================] - 3s 860ms/step - loss: 194.0288 - accuracy: 0.8125
Epoch 36/50
4/4 [==============================] - 3s 856ms/step - loss: 433.1166 - accuracy: 0.8021
Epoch 37/50
4/4 [==============================] - 4s 882ms/step - loss: 262.6776 - accuracy: 0.8021
Epoch 38/50
4/4 [==============================] - 3s 857ms/step - loss: 660.8314 - accuracy: 0.7292
Epoch 39/50
4/4 [==============================] - 3s 864ms/step - loss: 142.5472 - accuracy: 0.8750
Epoch 40/50
4/4 [==============================] - 3s 861ms/step - loss: 45.6001 - accuracy: 0.9375
Epoch 41/50
4/4 [==============================] - 3s 864ms/step - loss: 177.6790 - accuracy: 0.8750
Epoch 42/50
4/4 [==============================] - 3s 860ms/step - loss: 801.9673 - accuracy: 0.6042
Epoch 43/50
4/4 [==============================] - 4s 873ms/step - loss: 535.5167 - accuracy: 0.8021
Epoch 44/50
4/4 [==============================] - 3s 858ms/step - loss: 243.9561 - accuracy: 0.7604
Epoch 45/50
4/4 [==============================] - 3s 860ms/step - loss: 339.5434 - accuracy: 0.7917
Epoch 46/50
4/4 [==============================] - 4s 871ms/step - loss: 455.1973 - accuracy: 0.8125
Epoch 47/50
4/4 [==============================] - 3s 863ms/step - loss: 127.4080 - accuracy: 0.9375
Epoch 48/50
4/4 [==============================] - 3s 860ms/step - loss: 373.2668 - accuracy: 0.8021
Epoch 49/50
4/4 [==============================] - 3s 865ms/step - loss: 136.4450 - accuracy: 0.8542
Epoch 50/50
4/4 [==============================] - 3s 854ms/step - loss: 144.3659 - accuracy: 0.8542
```

The above image shows that after 50 Epochs and due to good accuracy, our model has fitted well with the high resolution Greek character dataset.

3.7.4 ResNet50 model with Low resolution Greek Character training dataset.



```
In [103]: model = ResNet50(include_top=False, weights='imagenet')
          x = model.output
          x = GlobalAveragePooling2D()(x)
          x = Dense(1024, activation='relu')(x)
          predictions = Dense(train_generator.num_classes, activation='softmax')(x)
          model_ResNet50_low = Model(inputs=model.input, outputs=predictions)

          for layer in model.layers:
              layer.trainable = False

          model_ResNet50_low.compile(optimizer='adam', loss='categorical_crossentropy', metrics = ['accuracy'])

          model_ResNet50_low.fit(train_generator, epochs = 15)
```

ResNet50 is an inbuilt model in TensorFlow which is used in the above image, ImageNet is specified as weights for this model, and ResNet50 works pretty well with the image dataset it is also easy to build.

The first activation function given is ReLu which takes only 1 and any positive number greater than 1, and if it is given 0 or any negative number as input it switches of the node.

The last activation function given is SoftMax which is explained in detail in the Activation Function subsection.

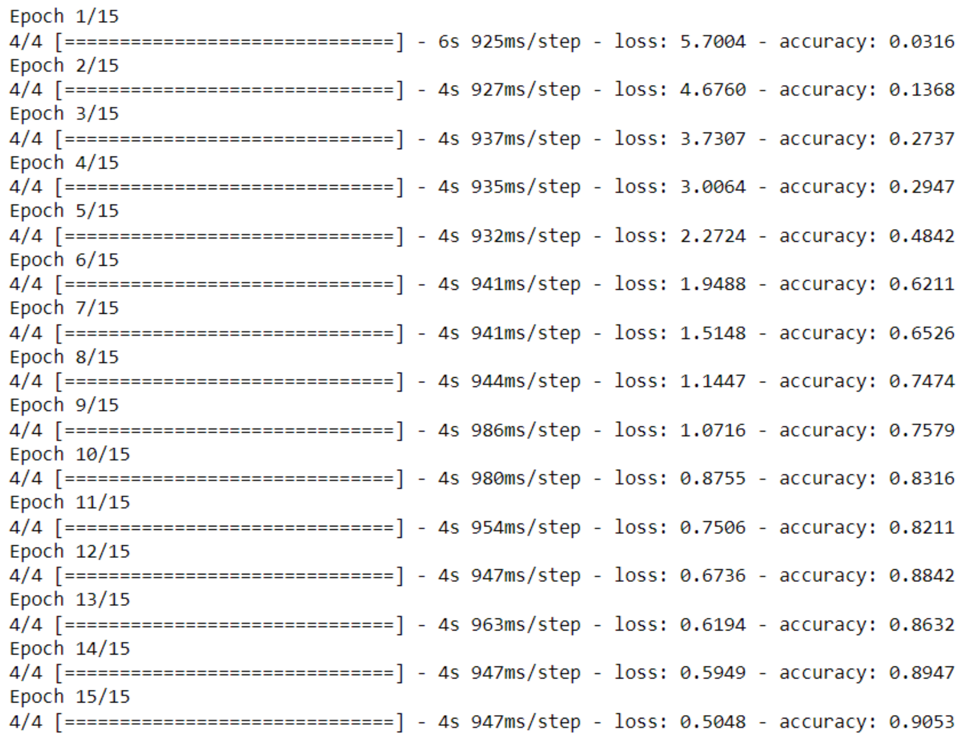
train_generator.num_classes specifies the number of total classes in the Handwritten Greek Character dataset, which is 24.

layer.trainable is initialized False to freeze the layer making it shift from trainable weights to non-trainable weights this helps a lot to get good accuracy.

The optimizers in deep learning are used for fine tuning the model to get good results on the test set, here the optimizer used is *Adam* which is the most commonly used to optimize. The learning rate is initialized as **0.0001** the smaller the learning rate is better for good accuracy. Epsilon is initialized to 1e-07, epsilon should be a small value as it is responsible to tackle the error raised due to dividing by zero.

model.compile function is used to compile the final specification that are the optimizer used, the loss that can be specified as **binary cross entropy or categorical cross entropy (which is used in this project).** Categorical cross entropy is used when we are dealing with Multi class image dataset.

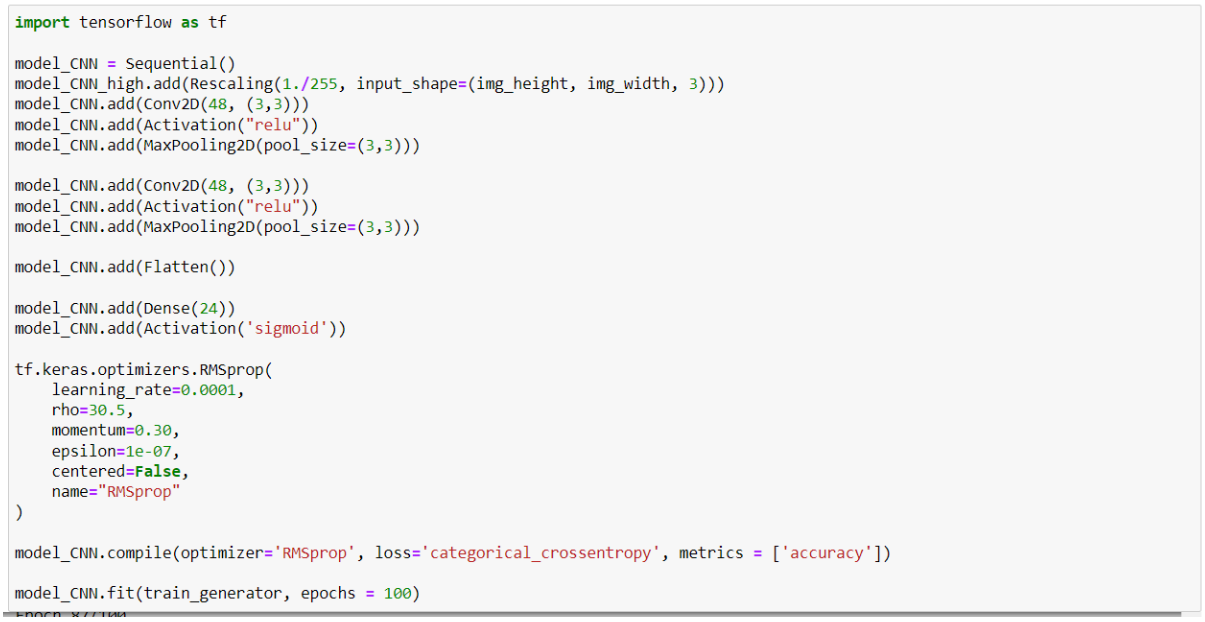
Then in the end the model created is fitted with train_generator as input and with 15 epochs with the help of model.fit() function.



```
Epoch 1/15
4/4 [==============================] - 6s 925ms/step - loss: 5.7004 - accuracy: 0.0316
Epoch 2/15
4/4 [==============================] - 4s 927ms/step - loss: 4.6760 - accuracy: 0.1368
Epoch 3/15
4/4 [==============================] - 4s 937ms/step - loss: 3.7307 - accuracy: 0.2737
Epoch 4/15
4/4 [==============================] - 4s 935ms/step - loss: 3.0064 - accuracy: 0.2947
Epoch 5/15
4/4 [==============================] - 4s 932ms/step - loss: 2.2724 - accuracy: 0.4842
Epoch 6/15
4/4 [==============================] - 4s 941ms/step - loss: 1.9488 - accuracy: 0.6211
Epoch 7/15
4/4 [==============================] - 4s 941ms/step - loss: 1.5148 - accuracy: 0.6526
Epoch 8/15
4/4 [==============================] - 4s 944ms/step - loss: 1.1447 - accuracy: 0.7474
Epoch 9/15
4/4 [==============================] - 4s 986ms/step - loss: 1.0716 - accuracy: 0.7579
Epoch 10/15
4/4 [==============================] - 4s 980ms/step - loss: 0.8755 - accuracy: 0.8316
Epoch 11/15
4/4 [==============================] - 4s 954ms/step - loss: 0.7506 - accuracy: 0.8211
Epoch 12/15
4/4 [==============================] - 4s 947ms/step - loss: 0.6736 - accuracy: 0.8842
Epoch 13/15
4/4 [==============================] - 4s 963ms/step - loss: 0.6194 - accuracy: 0.8632
Epoch 14/15
4/4 [==============================] - 4s 947ms/step - loss: 0.5949 - accuracy: 0.8947
Epoch 15/15
4/4 [==============================] - 4s 947ms/step - loss: 0.5048 - accuracy: 0.9053
```

The above image shows that after 15 Epochs and due to good accuracy, our model has fitted well with the low resolution Greek character dataset.

3.7.5 CNN model with Low resolution Greek Character training dataset.



```
import tensorflow as tf

model_CNN = Sequential()
model_CNN_high.add(Rescaling(1./255, input_shape=(img_height, img_width, 3)))
model_CNN.add(Conv2D(48, (3,3)))
model_CNN.add(Activation("relu"))
model_CNN.add(MaxPooling2D(pool_size=(3,3)))

model_CNN.add(Conv2D(48, (3,3)))
model_CNN.add(Activation("relu"))
model_CNN.add(MaxPooling2D(pool_size=(3,3)))

model_CNN.add(Flatten())

model_CNN.add(Dense(24))
model_CNN.add(Activation('sigmoid'))

tf.keras.optimizers.RMSprop(
    learning_rate=0.0001,
    rho=30.5,
    momentum=0.30,
    epsilon=1e-07,
    centered=False,
    name="RMSprop"
)

model_CNN.compile(optimizer='RMSprop', loss='categorical_crossentropy', metrics = ['accuracy'])

model_CNN.fit(train_generator, epochs = 100)
```

CNN is an inbuilt model in TensorFlow which has Conv2D, Activation, Maxpooling2D as the layers in tensorflow.keras.layers which are used in the above image.

Rescaling is used to process the image input, normalize it by dividing it with 255.0, and give the input shape as specified while pre-processing the image dataset with a dimension of 3 dim.

The first activation function given is ReLu which takes only 1 and any positive number greater than 1, and if it is given 0 or any negative number as input it switches of the node.

The Max pooling layer is responsible for the Suppression of the noise produced during the training.

The last activation function given is SoftMax which is explained in detail in the Activation Function subsection.

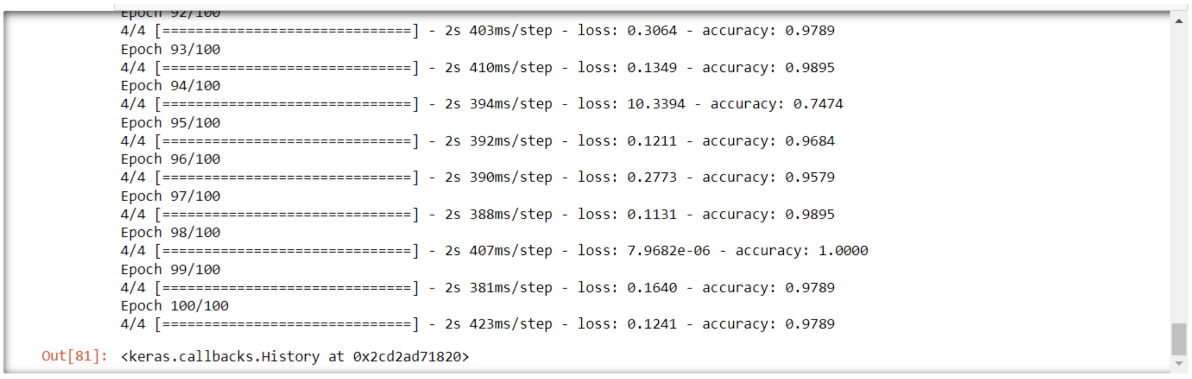
train_generator.num_classes specifies the number of total classes in the Handwritten Greek Character dataset, which is 24.

For CNN initializing layer.trainable to False doesn't work as the accuracy gets stuck to repeating the same accuracy again and again.

The optimizers in deep learning are used for fine tuning the model to get good results on the test set, here the optimizer used is *RmsProp* which is also the most commonly used to optimize. Rho is specified as **2.5** which is responsible for calculating the potential weighted average over the square of the gradients. The momentum is used to overcome the problem of noisy gradients which is specified to **0.20**. The learning rate is initialized as **0.001** the smaller the learning rate is better for good accuracy. Epsilon is initialized to 1e-07, epsilon should be a small value as it is responsible to tackle the error raised due to dividing by zero.

model.compile function is used to compile the final specification that are the optimizer used, the loss that can be specified as **binary cross entropy or categorical cross entropy (which is used in this project).** Categorical cross entropy is used when we are dealing with Multi class image dataset.

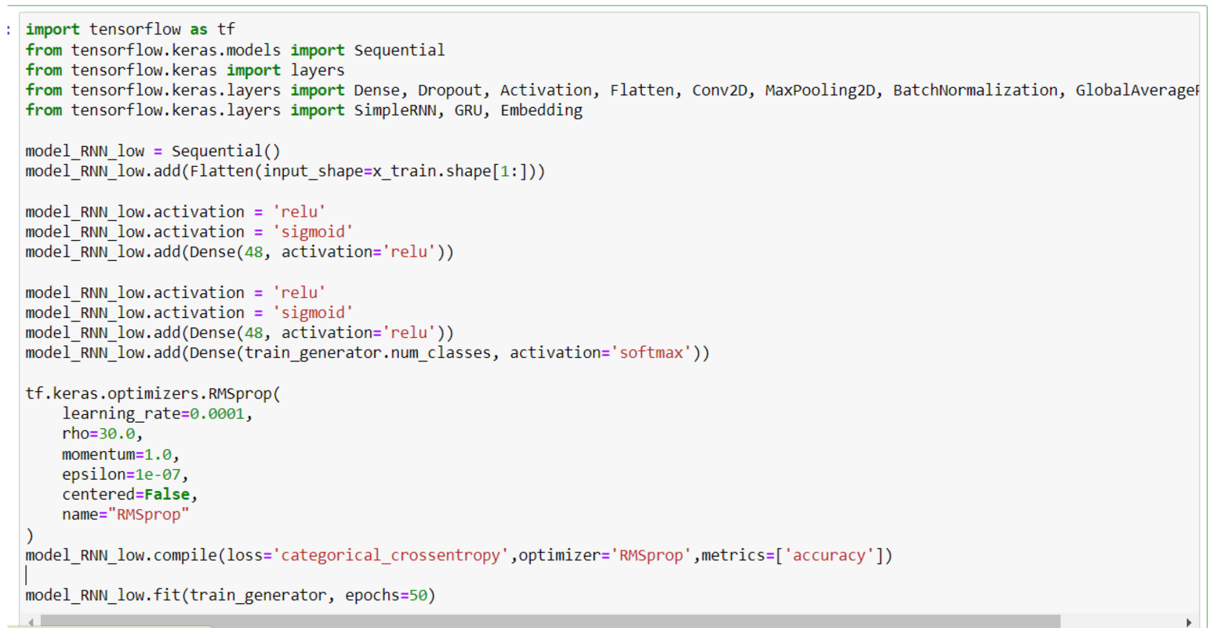
Then in the end the model created is fitted with train_generator as input and with 100 epochs with the help of model.fit() function.



```
Epoch 92/100
4/4 [==============================] - 2s 403ms/step - loss: 0.3064 - accuracy: 0.9789
Epoch 93/100
4/4 [==============================] - 2s 410ms/step - loss: 0.1349 - accuracy: 0.9895
Epoch 94/100
4/4 [==============================] - 2s 394ms/step - loss: 10.3394 - accuracy: 0.7474
Epoch 95/100
4/4 [==============================] - 2s 392ms/step - loss: 0.1211 - accuracy: 0.9684
Epoch 96/100
4/4 [==============================] - 2s 390ms/step - loss: 0.2773 - accuracy: 0.9579
Epoch 97/100
4/4 [==============================] - 2s 388ms/step - loss: 0.1131 - accuracy: 0.9895
Epoch 98/100
4/4 [==============================] - 2s 407ms/step - loss: 7.9682e-06 - accuracy: 1.0000
Epoch 99/100
4/4 [==============================] - 2s 381ms/step - loss: 0.1640 - accuracy: 0.9789
Epoch 100/100
4/4 [==============================] - 2s 423ms/step - loss: 0.1241 - accuracy: 0.9789

Out[81]: <keras.callbacks.History at 0x2cd2ad71820>
```

The above image shows that after 100 Epochs and due to good accuracy, our model has fitted well with the low resolution Greek character dataset.

3.7.6 RNN model with Low resolution Greek Character training dataset.



```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, BatchNormalization, GlobalAverageP
from tensorflow.keras.layers import SimpleRNN, GRU, Embedding

model_RNN_low = Sequential()
model_RNN_low.add(Flatten(input_shape=x_train.shape[1:]))

model_RNN_low.activation = 'relu'
model_RNN_low.activation = 'sigmoid'
model_RNN_low.add(Dense(48, activation='relu'))

model_RNN_low.activation = 'relu'
model_RNN_low.activation = 'sigmoid'
model_RNN_low.add(Dense(48, activation='relu'))
model_RNN_low.add(Dense(train_generator.num_classes, activation='softmax'))

tf.keras.optimizers.RMSprop(
    learning_rate=0.0001,
    rho=30.0,
    momentum=1.0,
    epsilon=1e-07,
    centered=False,
    name="RMSprop"
)
model_RNN_low.compile(loss='categorical_crossentropy',optimizer='RMSprop',metrics=['accuracy'])

model_RNN_low.fit(train_generator, epochs=50)
```

Here RNN is built from scratch and TensorFlow has Dense and Flatten as the layers in tensorflow.keras.layers which are used in the above image.

The first activation function given is ReLu which takes only 1 and any positive number greater than 1, and if it is given 0 or any negative number as input it switches of the node.

Flatten is used to manipulate the input shape.

The Sigmoid function is also an activation function used to convert the input into probability which can also be stated as mapping the values into very small numbers that are between 0 to 1 or -1 to 1

The last activation function given is SoftMax which is explained in detail in the Activation Function subsection.

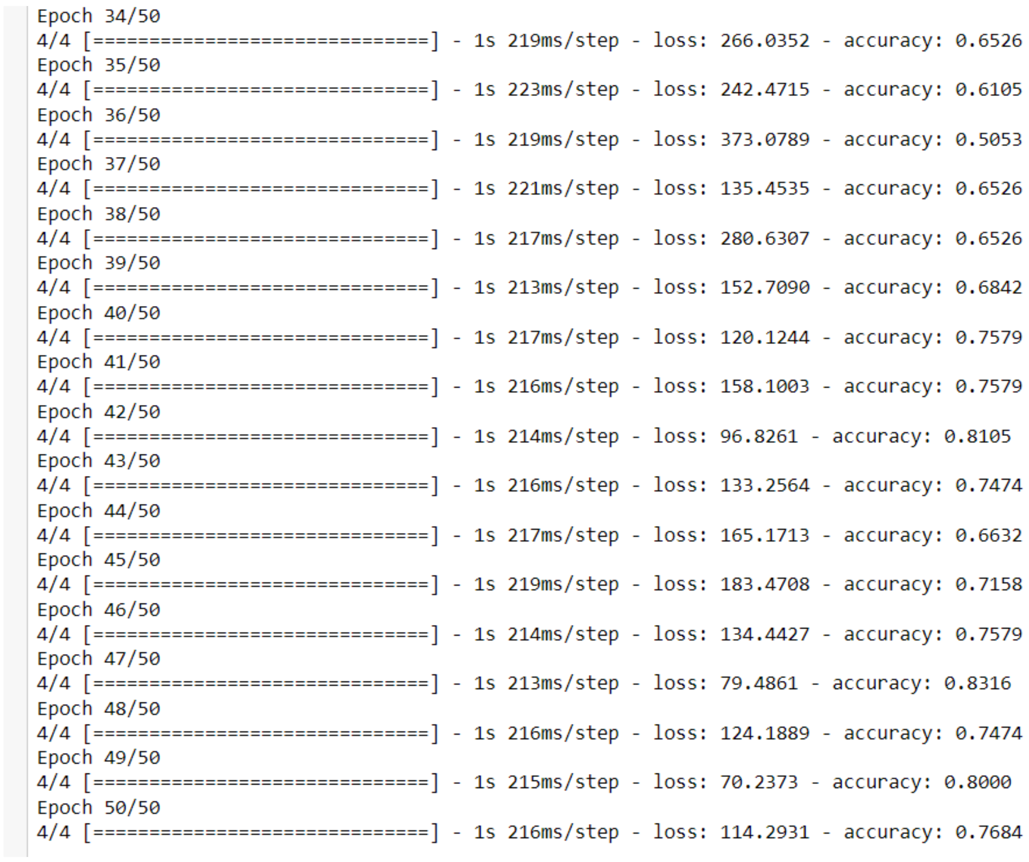
train_generator.num_classes specifies the number of total classes in the Handwritten Greek Character dataset, which is 24.

For RNN initializing layer.trainable to False doesn't work as the accuracy gets stuck to repeating the same accuracy again and again.

The optimizers in deep learning are used for fine tuning the model to get good results on the test set, here the optimizer used is *RmsProp* which is also the most commonly used to optimize. Rho is specified as **30.0** which is responsible for calculating the potential weighted average over the square of the gradients. The momentum is used to overcome the problem of noisy gradients which is specified to **1.0**. The learning rate is initialized as **0.001** the smaller the learning rate is better for good accuracy. Epsilon is initialized to **1e-07**, epsilon should be a small value as it is responsible to tackle the error raised due to dividing by zero.

model.compile function is used to compile the final specification that are the optimizer used, the loss that can be specified as **binary cross entropy or categorical cross entropy (which is used in this project).** Categorical cross entropy is used when we are dealing with Multi class image dataset.

Then in the end the model created is fitted with train_generator as input and with 50 epochs with the help of model.fit() function.

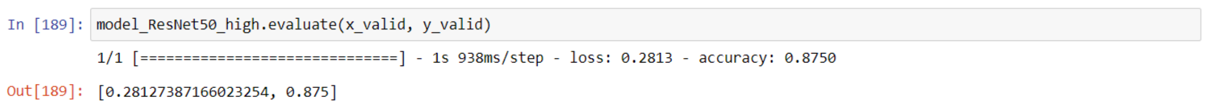


```
Epoch 34/50
4/4 [==============================] - 1s 219ms/step - loss: 266.0352 - accuracy: 0.6526
Epoch 35/50
4/4 [==============================] - 1s 223ms/step - loss: 242.4715 - accuracy: 0.6105
Epoch 36/50
4/4 [==============================] - 1s 219ms/step - loss: 373.0789 - accuracy: 0.5053
Epoch 37/50
4/4 [==============================] - 1s 221ms/step - loss: 135.4535 - accuracy: 0.6526
Epoch 38/50
4/4 [==============================] - 1s 217ms/step - loss: 280.6307 - accuracy: 0.6526
Epoch 39/50
4/4 [==============================] - 1s 213ms/step - loss: 152.7090 - accuracy: 0.6842
Epoch 40/50
4/4 [==============================] - 1s 217ms/step - loss: 120.1244 - accuracy: 0.7579
Epoch 41/50
4/4 [==============================] - 1s 216ms/step - loss: 158.1003 - accuracy: 0.7579
Epoch 42/50
4/4 [==============================] - 1s 214ms/step - loss: 96.8261 - accuracy: 0.8105
Epoch 43/50
4/4 [==============================] - 1s 216ms/step - loss: 133.2564 - accuracy: 0.7474
Epoch 44/50
4/4 [==============================] - 1s 217ms/step - loss: 165.1713 - accuracy: 0.6632
Epoch 45/50
4/4 [==============================] - 1s 219ms/step - loss: 183.4708 - accuracy: 0.7158
Epoch 46/50
4/4 [==============================] - 1s 214ms/step - loss: 134.4427 - accuracy: 0.7579
Epoch 47/50
4/4 [==============================] - 1s 213ms/step - loss: 79.4861 - accuracy: 0.8316
Epoch 48/50
4/4 [==============================] - 1s 216ms/step - loss: 124.1889 - accuracy: 0.7474
Epoch 49/50
4/4 [==============================] - 1s 215ms/step - loss: 70.2373 - accuracy: 0.8000
Epoch 50/50
4/4 [==============================] - 1s 216ms/step - loss: 114.2931 - accuracy: 0.7684
```

The above image shows that after 50 Epochs and due to good accuracy, our model has fitted well with the low resolution Greek character dataset.

3.8 Evaluating

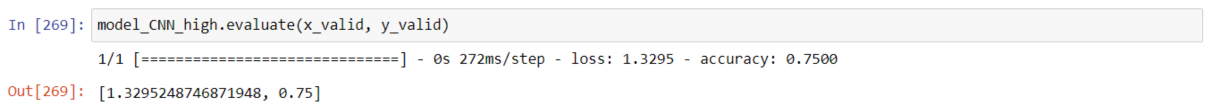
3.8.1 ResNet50 model with High resolution Greek Character validation dataset.



```
In [189]: model_ResNet50_high.evaluate(x_valid, y_valid)
          1/1 [==============================] - 1s 938ms/step - loss: 0.2813 - accuracy: 0.8750
Out[189]: [0.28127387166023254, 0.875]
```

After evaluating the ResNet50 model which is trained on high resolution Greek character training dataset on the validation dataset is 0.8750 which is a pretty good score, and we can move forward to test it on the test dataset.

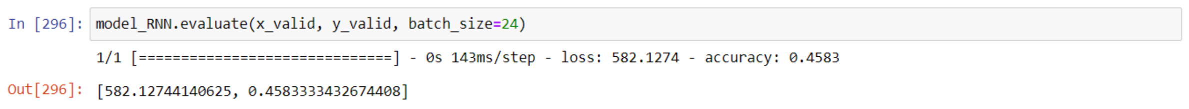
3.8.2 CNN model with High resolution Greek Character validation dataset.



```
In [269]: model_CNN_high.evaluate(x_valid, y_valid)
          1/1 [==============================] - 0s 272ms/step - loss: 1.3295 - accuracy: 0.7500
Out[269]: [1.3295248746871948, 0.75]
```

After evaluating the CNN model which is trained on high resolution Greek character training dataset on the validation dataset is 0.7500 which is a pretty good score, and we can move forward to test it on the test dataset.

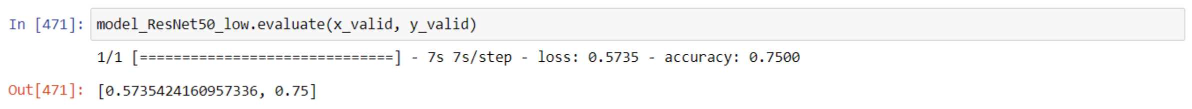
3.8.3 RNN model with High resolution Greek Character validation dataset.



```
In [296]: model_RNN.evaluate(x_valid, y_valid, batch_size=24)
          1/1 [==============================] - 0s 143ms/step - loss: 582.1274 - accuracy: 0.4583
Out[296]: [582.12744140625, 0.4583333432674408]
```

After evaluating the RNN model which is trained on high resolution Greek character training dataset on the validation dataset is 0.4583 which is not a good score, but RNN is difficult to work with image datasets as it is more complex. We can move forward to test it on the test dataset.

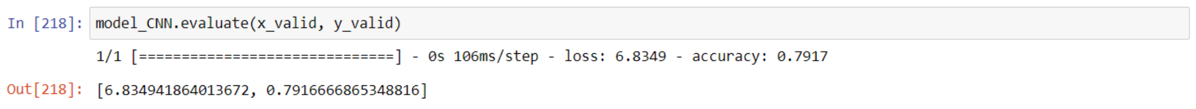
3.8.4 ResNet50 model with Low resolution Greek Character validation dataset.



```
In [471]: model_ResNet50_low.evaluate(x_valid, y_valid)
          1/1 [==============================] - 7s 7s/step - loss: 0.5735 - accuracy: 0.7500
Out[471]: [0.5735424160957336, 0.75]
```

After evaluating the ResNet50 model which is trained on low resolution Greek character training dataset on the validation dataset is 0.7500 which is a pretty good score, and we can move forward to test it on the test dataset.

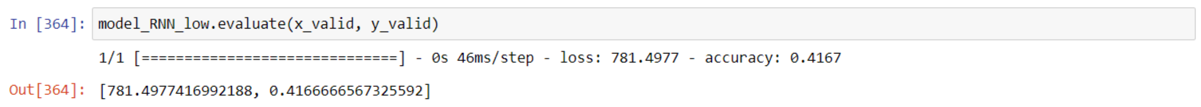
3.8.5 CNN model with Low resolution Greek Character validation dataset.



```
In [218]: model_CNN.evaluate(x_valid, y_valid)
          1/1 [==============================] - 0s 106ms/step - loss: 6.8349 - accuracy: 0.7917
Out[218]: [6.834941864013672, 0.7916666865348816]
```

After evaluating the CNN model which is trained on low resolution Greek character training dataset on the validation dataset is 0.7917 which is a pretty good score, and we can move forward to test it on the test dataset.

3.8.6 RNN model with Low resolution Greek Character validation dataset.

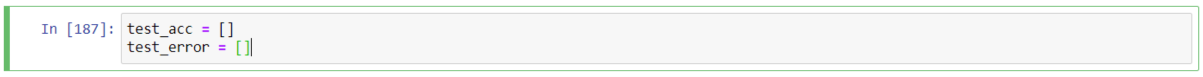


```
In [364]: model_RNN_low.evaluate(x_valid, y_valid)

          1/1 [==============================] - 0s 46ms/step - loss: 781.4977 - accuracy: 0.4167

Out[364]: [781.4977416992188, 0.4166666567325592]
```

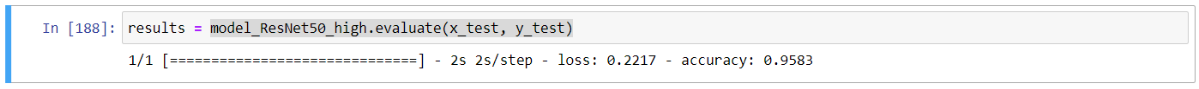
After evaluating the RNN model which is trained on low resolution Greek character training dataset on the validation dataset is 0.4167 which is not that good score, but we can move forward to test it on the test dataset.

3.9 Testing

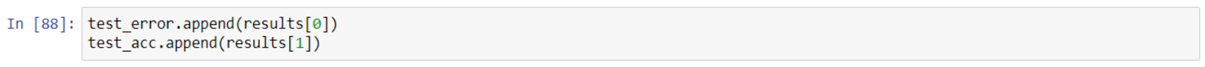
3.9.1 ResNet50 model with High resolution Greek Character Test dataset.



```
In [187]: test_acc = []
          test_error = []
```



```
In [188]: results = model_ResNet50_high.evaluate(x_test, y_test)

          1/1 [==============================] - 2s 2s/step - loss: 0.2217 - accuracy: 0.9583
```



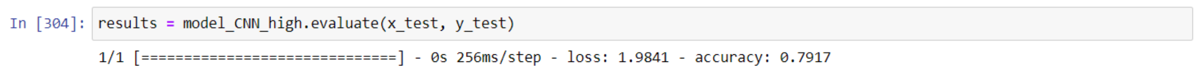
```
In [88]: test_error.append(results[0])
         test_acc.append(results[1])
```

test_acc and test_error are the variables used for making a list of all the accuracies and all the errors obtained of each model and then visualize them with the help of these lists.

The loss and accuracy for model_ResNet50_high is 0.9583 which is pretty good for the model we will test this accuracy in the prediction subsection for more confidence and detail with visualization.

results variable is used further for plotting bar and line plots.

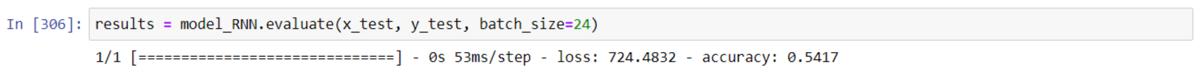
3.9.2 CNN model with High resolution Greek Character Test dataset.



```
In [304]: results = model_CNN_high.evaluate(x_test, y_test)

          1/1 [==============================] - 0s 256ms/step - loss: 1.9841 - accuracy: 0.7917
```

As we saw in the Evaluating section CNN model on the validation set for high resolution Greek character test got an accuracy of 0.7500 and here accuracy for the same on the test set, we got an accuracy of 0.7917 which is a good change and indicates that our model is working well.

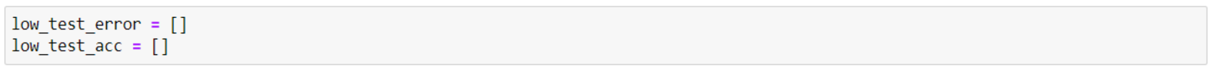
3.9.3 RNN model with High resolution Greek Character Test dataset.



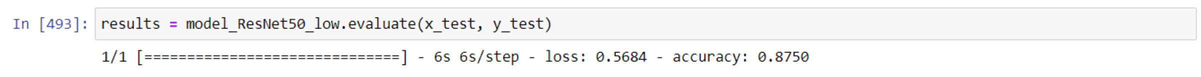
```
In [306]: results = model_RNN.evaluate(x_test, y_test, batch_size=24)

          1/1 [==============================] - 0s 53ms/step - loss: 724.4832 - accuracy: 0.5417
```

As we saw in the Evaluating section RNN model on the validation set for high resolution Greek character test got an accuracy of 0.4583 and here accuracy for the same on the test set we got an accuracy of 0.5417 which is a good change and but still not a good score compared to ResNet50 and CNN.

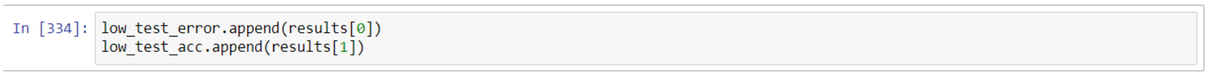
3.9.4 ResNet50 model with Low resolution Greek Character Test dataset.



```
low_test_error = []
low_test_acc = []
```



```
In [493]: results = model_ResNet50_low.evaluate(x_test, y_test)

          1/1 [==============================] - 6s 6s/step - loss: 0.5684 - accuracy: 0.8750
```

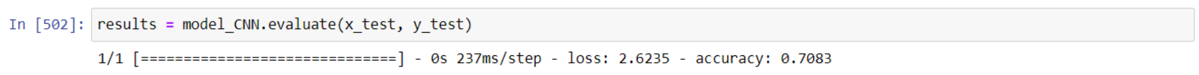


```
In [334]: low_test_error.append(results[0])
          low_test_acc.append(results[1])
```

The same thing we did with the High Resolution Greek character dataset we will follow the same steps making 2 lists for low resolution Greek character dataset errors and accuracies.

Here we got an accuracy of 0.8750 for test dataset which is more than the accuracy we got for the validation set which is a significant change and also a good result, we will test this accuracy in the prediction subsection for more confidence and detail with visualization.

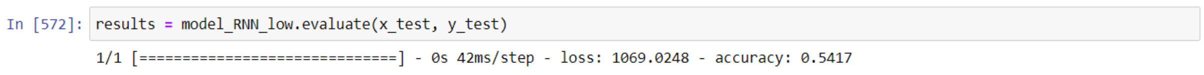
3.9.5 CNN model with Low resolution Greek Character Test dataset.



```
In [502]: results = model_CNN.evaluate(x_test, y_test)

          1/1 [==============================] - 0s 237ms/step - loss: 2.6235 - accuracy: 0.7083
```

As we saw in the Evaluating section CNN model on the validation set for low resolution Greek character dataset, we got an accuracy of 0. 0.7917 and here we got an accuracy of 0.7083 which is not a good change, but we can test that in the prediction and visualization section.

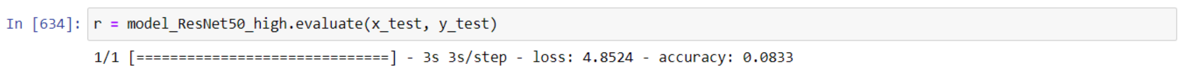
3.9.6 RNN model with Low resolution Greek Character Test dataset.



```
In [572]: results = model_RNN_low.evaluate(x_test, y_test)

          1/1 [==============================] - 0s 42ms/step - loss: 1069.0248 - accuracy: 0.5417
```

As we saw in the Evaluating section RNN model on the validation set for low resolution Greek character test got an accuracy of 0.4167 and here accuracy for the same on the test set we got an accuracy of 0.5417 which is a good change and but still not a good score compared to ResNet50 and CNN.

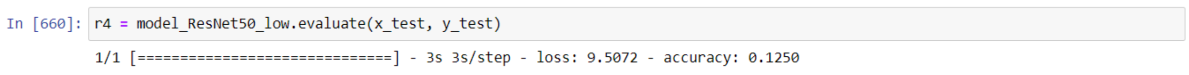
3.9.7 ResNet50 trained model on High resolution Greek Characters with Low resolution Test dataset.



```
In [634]: r = model_ResNet50_high.evaluate(x_test, y_test)

          1/1 [==============================] - 3s 3s/step - loss: 4.8524 - accuracy: 0.0833
```

This experiment got some interesting accuracies for example here for ResNet50 model which is trained on High resolution Greek character dataset got very low accuracy when tested on the low resolution Greek character dataset.

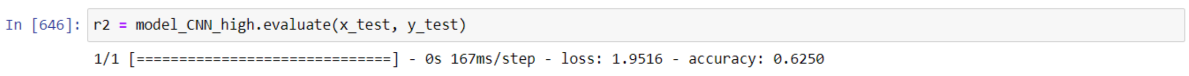
3.9.8 ResNet50 trained model on Low resolution Greek Characters with High resolution Test dataset.



```
In [660]: r4 = model_ResNet50_low.evaluate(x_test, y_test)

          1/1 [==============================] - 3s 3s/step - loss: 9.5072 - accuracy: 0.1250
```

Here for ResNet50 model which is trained on Low resolution Greek character dataset got very low accuracy when tested on the high resolution Greek character dataset.

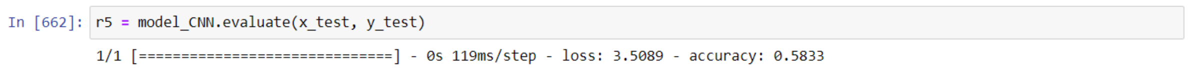
3.9.9 CNN trained model on High resolution Greek Characters with Low resolution Test dataset.



```
In [646]: r2 = model_CNN_high.evaluate(x_test, y_test)

          1/1 [==============================] - 0s 167ms/step - loss: 1.9516 - accuracy: 0.6250
```

Here for CNN model which is trained on High resolution Greek character dataset got pretty good accuracy when tested on the low resolution Greek character dataset. As getting this accuracy is a very different thing as previously seen CNN was not better than ResNet50 model but now CNN appears to be the best when opposite datasets are used to train and test datasets.

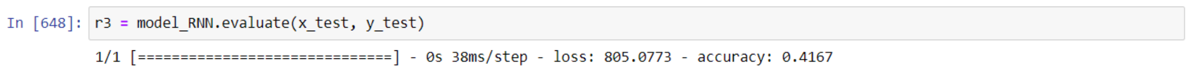
3.9.10 CNN trained model on Low resolution Greek Characters with High resolution Test dataset.



```
In [662]: r5 = model_CNN.evaluate(x_test, y_test)

          1/1 [==============================] - 0s 119ms/step - loss: 3.5089 - accuracy: 0.5833
```

Here for CNN model which is trained on Low resolution Greek character dataset got pretty good accuracy when tested on the high resolution Greek character dataset. As getting this accuracy is a very different thing as previously seen CNN was not better than ResNet50 model but now CNN appears to be the best when opposite datasets are used to train and test datasets.

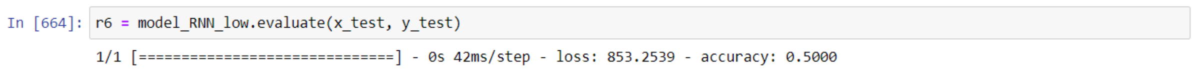
3.9.11 RNN trained model on High resolution Greek Characters with Low resolution Test dataset.



```
In [648]: r3 = model_RNN.evaluate(x_test, y_test)

          1/1 [==============================] - 0s 38ms/step - loss: 805.0773 - accuracy: 0.4167
```

Here for RNN model which is trained on High resolution Greek character dataset got pretty good accuracy when tested on the low resolution Greek character dataset, which was not expected at all seen its history of accuracy when compared to ResNet50 model and CNN model, but now RNN appears to be the better than ResNet50 when opposite datasets are used to train and test datasets.

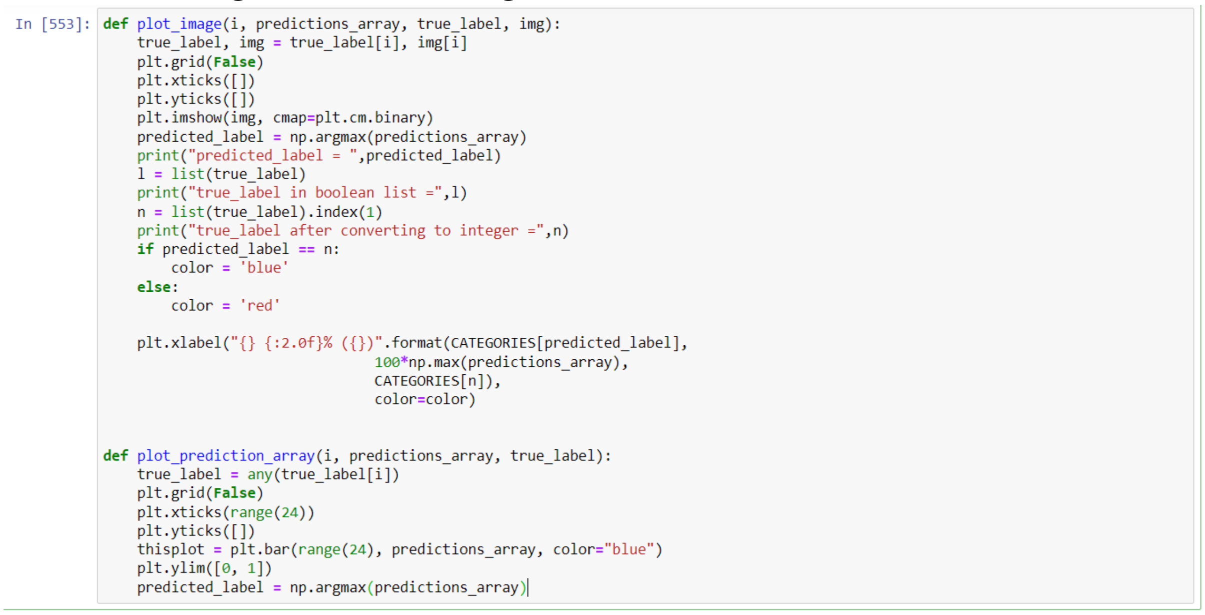
3.9.12 RNN trained model on Low resolution Greek Characters with High resolution Test dataset.



```
In [664]: r6 = model_RNN_low.evaluate(x_test, y_test)

          1/1 [==============================] - 0s 42ms/step - loss: 853.2539 - accuracy: 0.5000
```

Here for RNN model which is trained on Low resolution Greek character dataset got pretty good accuracy when tested on the high resolution Greek character dataset, which was not expected at all seen its history of accuracy when compared to ResNet50 model and CNN model, but now RNN appears to be the better than ResNet50 when opposite datasets are used to train and test datasets.

3.10 Predicting and Visualizing



```
In [553]: def plot_image(i, predictions_array, true_label, img):
              true_label, img = true_label[i], img[i]
              plt.grid(False)
              plt.xticks([])
              plt.yticks([])
              plt.imshow(img, cmap=plt.cm.binary)
              predicted_label = np.argmax(predictions_array)
              print("predicted_label = ",predicted_label)
              l = list(true_label)
              print("true_label in boolean list =",l)
              n = list(true_label).index(1)
              print("true_label after converting to integer =",n)
              if predicted_label == n:
                  color = 'blue'
              else:
                  color = 'red'

              plt.xlabel("{} {:2.0f}% ({})".format(CATEGORIES[predicted_label],
                                          100*np.max(predictions_array),
                                          CATEGORIES[n]),
                                          color=color)


          def plot_prediction_array(i, predictions_array, true_label):
              true_label = any(true_label[i])
              plt.grid(False)
              plt.xticks(range(24))
              plt.yticks([])
              thisplot = plt.bar(range(24), predictions_array, color="blue")
              plt.ylim([0, 1])
              predicted_label = np.argmax(predictions_array)
```

In the above snippet of code there are two functions used one to plot the image which we will be predicting and one for plotting the prediction with a bar plot. This code remains the same for all the predictions. This piece of code is inspired for the official website of TensorFlow (TensorFlow, Basic classification: Classify images of clothing, 2017).

plot_image function takes 4 arguments “i” which is the number of the image we want to predict, here “i” ranges from 0 to 23 as there are 24 classes in the Greek characters dataset, prediction_array is used to call the predict function, true_label is the actual label which we will use to compare with the newly predicted label to see how right or how wrong is our model, and lastly image is the image we want to predict the label for.

The main thing here is the true_label which for us is in the form of a Boolean list to compare the true_label with the predicted label we first have to convert the Boolean list into an integer, this is done using list(true_label).index(1) this will count the position where 1 appears in the true_label’s list and give its position as an integer which will be further used to compare with the predicted_label.

If condition is used to see if predicted_label is equal to the true_label and if it’s true, then the colour of the predicted label and the true label will be blue or else it will be red.

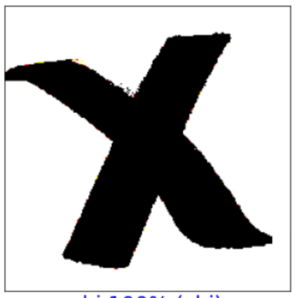
plt.xlabel is used to give a structure of how the true label and the predicted label will be printed. True label is taken from the CATEGORIES list which is initialized in the initializing in the categories section.

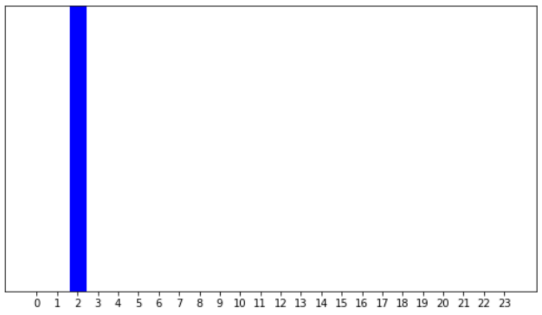
Next comes the plot_prediction_array function which is responsible for plotting the prediction bar plot. This bar plot has 24 columns starting from 0 to 23. Blue coloured bar will represent the predicted label and orange colour bar will represent the true label.

Let’s call both function and visualise it.

3.10.1 ResNet50 model with High resolution Greek Character Test dataset.

```
In [384]: import numpy as np
          i = 0
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions[i], y_test)
          plt.show()

          Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

          predicted_label =  2
          true_label in boolean list = [0.0, 0.0, 1.0, 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, 0.0, 0.0, 0.0]
          true_label after converting to integer = 2
```

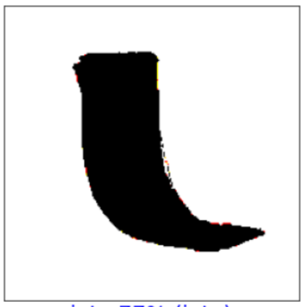

chi 100% (chi)

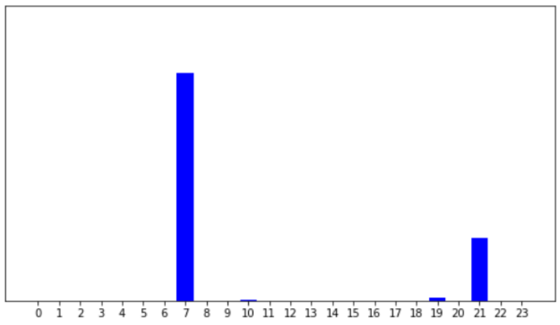


Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 0th image in this case is "chi" and as per the predict function and the ResNet50 model chi is predicted with 100% confidence as the predicted_label and the true label are equal to each other thus they are written in "blue" colour the bar plot shows 2 as the predicted label which is right.

chi 100% represents the predicted label and (chi) represents the true label.

```
In [710]: import numpy as np
          i = 23
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions[i], y_test)
          plt.show()

          Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

          predicted_label =  7
          true_label in boolean list = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 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]
          true_label after converting to integer = 7
```


iota 77% (iota)



Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 23rd image in this case is "iota" and as per the predict function and the ResNet50 model iota is predicted with 77% confidence as the predicted_label and the true label are equal to each other thus they are written in "blue" colour the bar plot shows 7 as the predicted label with highest confidence level compared to other bars which is predicted right.

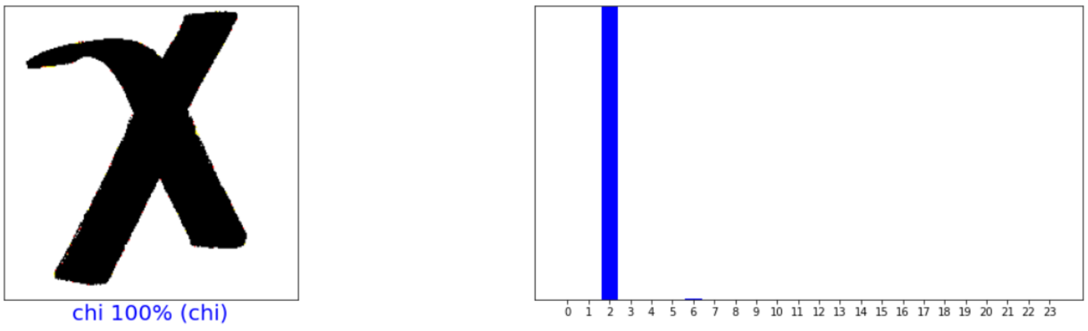
iota 77% represents the predicted label and (iota) represents the true label.

3.10.2 CNN model with High resolution Greek Character Test dataset.

```
In [712]: import numpy as np
          i = 0
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions_CNN_high[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions_CNN_high[i], y_test)
          plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
predicted_label =  2
true_label in boolean list = [0.0, 0.0, 1.0, 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, 0.0, 0.0, 0.0]
true_label after converting to integer = 2
```



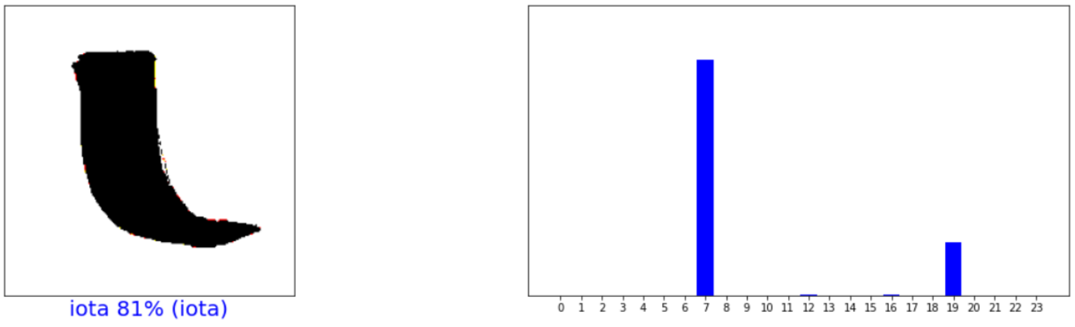
Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 0th image in this case is "chi" and as per the predict function and the CNN model chi is predicted with 100% confidence as the predicted_label and the true label are equal to each other thus they are written in "blue" colour the bar plot shows 2 as the predicted label which is right.

chi 100% represents the predicted label and (chi) represents the true label.

```
In [713]: import numpy as np
          i = 23
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions_CNN_high[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions_CNN_high[i], y_test)
          plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
predicted_label =  7
true_label in boolean list = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 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]
true_label after converting to integer = 7
```

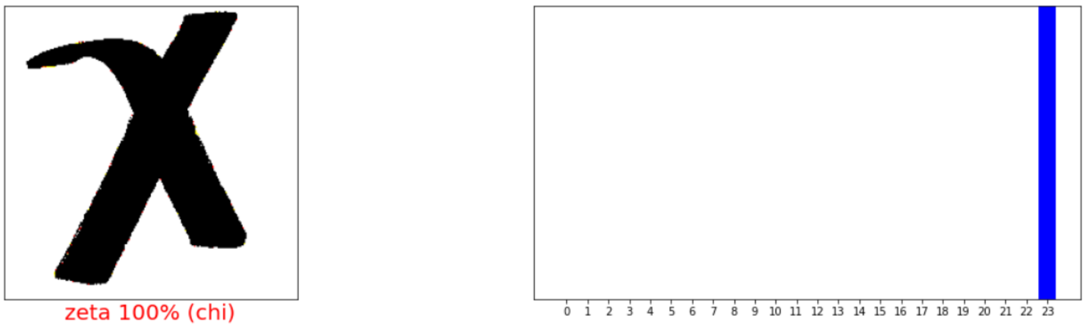


Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 23rd image in this case is "iota" and as per the predict function and the CNN model iota is predicted with 81% confidence as the predicted_label and the true label are equal to each other thus they are written in "blue" colour the bar plot shows 7 as the predicted label with highest confidence level compared to other bars which is predicted right.

iota 81% represents the predicted label and (iota) represents the true label.

3.10.3 RNN model with High resolution Greek Character Test dataset.

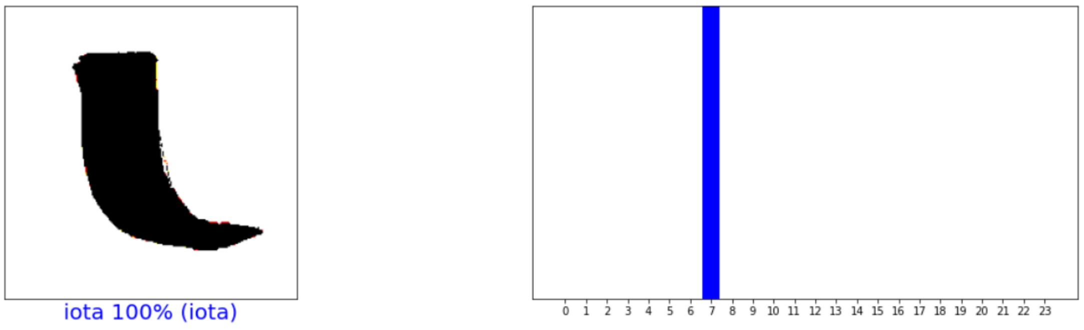
```
In [715]: import numpy as np
          i = 0
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions_RNN[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions_RNN[i], y_test)
          plt.show()

          Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

          predicted_label =  23
          true_label in boolean list = [0.0, 0.0, 1.0, 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, 0.0, 0.0, 0.0]
          true_label after converting to integer = 2
```



Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 0th image in this case is "chi" and as per the predict function and the RNN model zeta is predicted with 100% confidence as the predicted_label and the true label are not equal to each other they are written in "red" colour the bar plot shows 23 as the predicted label which is wrong.

zeta 100% represents the predicted label and (chi) represents the true label.

```
In [716]: import numpy as np
          i = 23
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions_RNN[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions_RNN[i], y_test)
          plt.show()

          Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

          predicted_label =  7
          true_label in boolean list = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 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]
          true_label after converting to integer = 7
```



Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 23rd image in this case is "iota" and as per the predict function and the RNN model iota is predicted with 100% confidence as the predicted_label and the true label are equal to each other thus they are written in "blue" colour the bar plot shows 7 as the predicted label with highest confidence level compared to other bars which is predicted right. Surprisingly RNN has predicted iota with 100% of confidence.

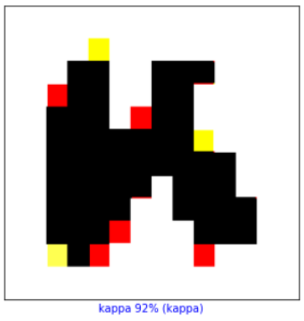
iota 100% represents the predicted label and (iota) represents the true label.

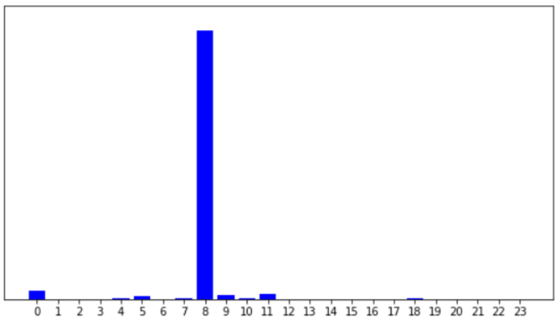
3.10.4 ResNet50 model with Low resolution Greek Character Test dataset.

```
In [774]: import numpy as np
          i = 0
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions[i], y_test)
          plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
predicted_label =  8
true_label in boolean list = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 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]
true_label after converting to integer = 8
```





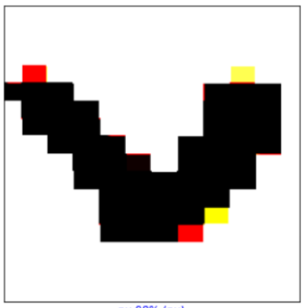
Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 0th image in this case is "kappa" and as per the predict function and the ResNet50 model kappa is predicted with 92% confidence as the predicted_label and the true label are equal to each other thus they are written in "blue" colour the bar plot shows 8 as the predicted label with the highest confidence level compared to other bars which is a good and a right result.

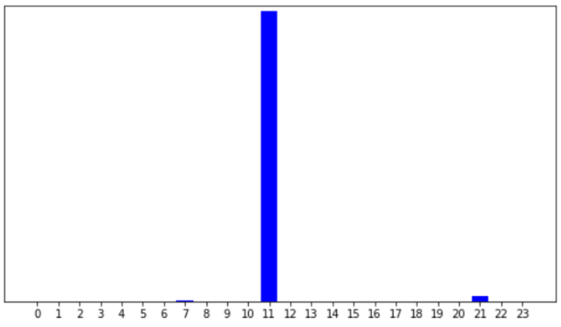
kappa 92% represents the predicted label and (kappa) represents the true label.

```
In [775]: import numpy as np
          i = 23
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions[i], y_test)
          plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
predicted_label =  11
true_label in boolean list = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0]
true_label after converting to integer = 11
```





Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 23rd image in this case is "nu" and as per the predict function and the ResNet50 model iota is predicted with 98% confidence as the predicted_label and the true label are equal to each other thus they are written in "blue" colour the bar plot shows 11 as the predicted label with highest confidence level compared to other bars which is predicted right.

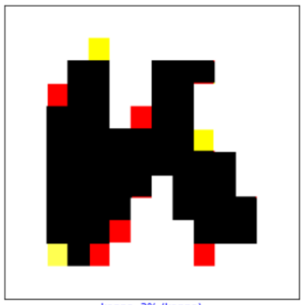
nu 98% represents the predicted label and (nu) represents the true label.

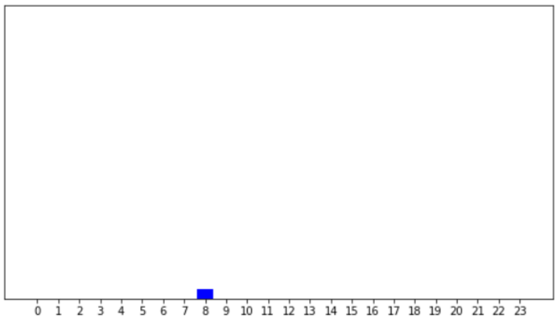
3.10.5 CNN model with Low resolution Greek Character Test dataset.

```
In [777]: import numpy as np
          i = 0
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions_CNN_low[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions_CNN_low[i], y_test)
          plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
predicted_label =  8
true_label in boolean list = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 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]
true_label after converting to integer = 8
```





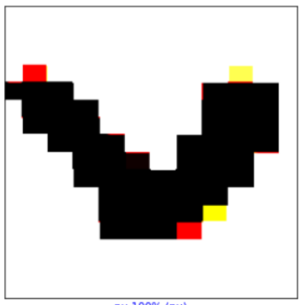
Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 0th image in this case is "kappa" and as per the predict function and the CNN model kappa is predicted with 3% confidence as the predicted_label and the true label are equal to each other thus they are written in "blue" colour the bar plot shows 8 with less confidence which is concerning though it has predicted the label right.

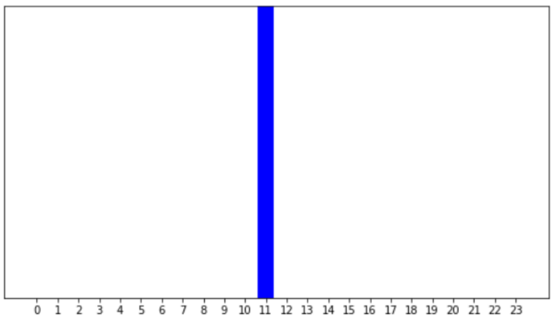
kappa 3% represents the predicted label and (kappa) represents the true label.

```
In [778]: import numpy as np
          i = 23
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions_CNN_low[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions_CNN_low[i], y_test)
          plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
predicted_label =  11
true_label in boolean list = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0]
true_label after converting to integer = 11
```


nu 100% (nu)



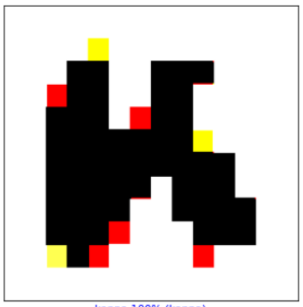
Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 23rd image in this case is "nu" and as per the predict function and the ResNet50 model iota is predicted with 100% confidence as the predicted_label and the true label are equal to each other thus they are written in "blue" colour the bar plot shows 11 as the predicted label with highest confidence level which is predicted right.

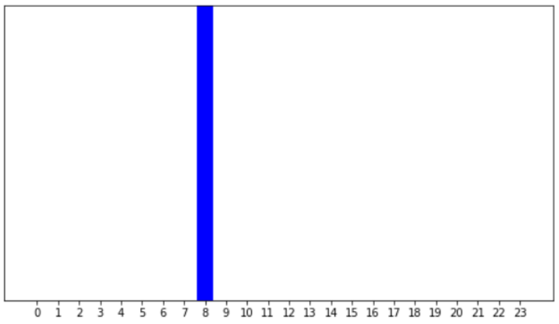
nu 100% represents the predicted label and (nu) represents the true label.

3.10.6 RNN model with Low resolution Greek Character Test dataset.

```
In [780]: import numpy as np
          i = 0
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions_RNN_low[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions_RNN_low[i], y_test)
          plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

predicted_label =  8
true_label in boolean list = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 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]
true_label after converting to integer = 8
```


kappa 100% (kappa)

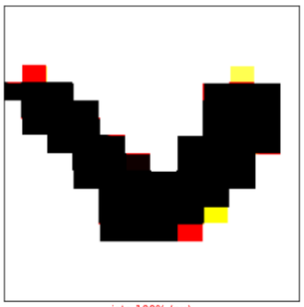


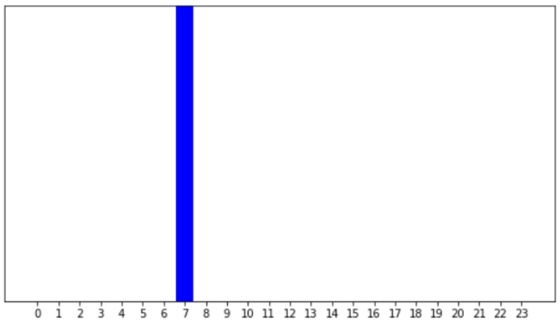
Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 0th image in this case is "kappa" and as per the predict function and the RNN model kappa is predicted with 100% confidence as the predicted_label and the true label are equal to each other thus they are written in "blue" colour the bar plot shows 8 as the predicted label with the highest confidence level which is a good and a right result.

kappa 100% represents the predicted label and (kappa) represents the true label.

```
In [781]: import numpy as np
          i = 23
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions_RNN_low[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions_RNN_low[i], y_test)
          plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

predicted_label =  7
true_label in boolean list = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0]
true_label after converting to integer = 11
```


iota 100% (nu)



Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 23rd image in this case is "nu" and as per the predict function and the RNN model iota is predicted with 100% confidence as the predicted_label and the true label are not equal to each other they are written in "red" colour, the bar plot shows 7 as the predicted label which is wrong because the true label of nu is 11.

iota 100% represents the predicted label and (nu) represents the true label.

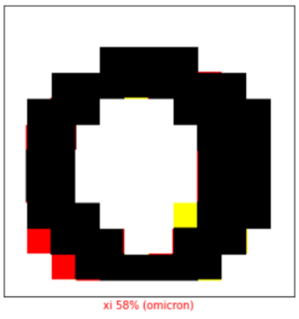
3.10.7 ResNet50 trained model on High resolution Greek Characters with Low resolution Test dataset.

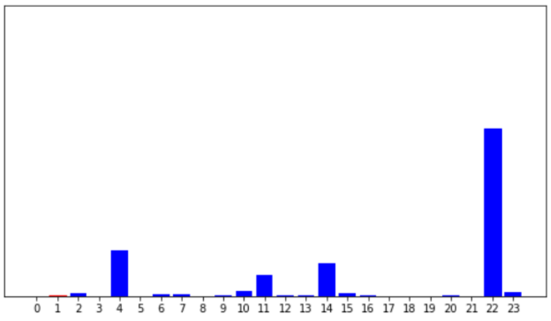
```
In [637]: predictions_1 = model_ResNet50_high.predict(x_test)
```

```
In [642]: import numpy as np
          i = 23
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions_1[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions_1[i], y_test)
          plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
predicted_label =  22
true_label in boolean list = [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, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0]
true_label after converting to integer = 13
```





As we have seen in the testing subsection "ResNet50 model which is trained on High resolution Greek character dataset got very low accuracy when tested on the low resolution Greek character dataset". Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 23rd image in this case is "omicron" and as per the predict function and the ResNet50 model xi is predicted with 58% confidence as the predicted_label and the true label are not equal to each other they are written in "red" colour, the bar plot shows 22 as the predicted label which is wrong because the true label of omicron is 13.

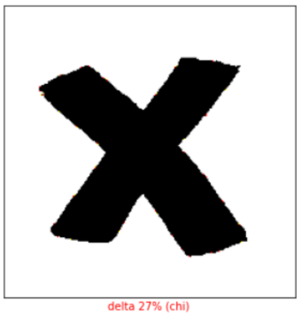
xi 58% represents the predicted label and (omicron) represents the true label.

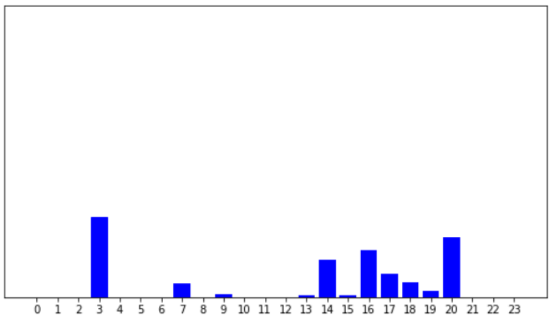
3.10.8 ResNet50 trained model on Low resolution Greek Characters with High resolution Test dataset.

```
In [661]: predictions_4 = model_ResNet50_low.predict(x_test)

          import numpy as np
          i = 23
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions_4[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions_4[i], y_test)
          plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
predicted_label =  3
true_label in boolean list = [0.0, 0.0, 1.0, 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, 0.0, 0.0, 0.0]
true_label after converting to integer = 2
```





As we have seen in the testing subsection "ResNet50 model which is trained on Low resolution Greek character dataset got very low accuracy when tested on the high resolution Greek character dataset". Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 23rd image in this case is "chi" and as per the predict function and the ResNet50 model delta is predicted with 27% confidence as the predicted_label and the true label are not equal to each other they are written in "red" colour, the bar plot shows 3 as the predicted label which is wrong because the true label of chi is 2.

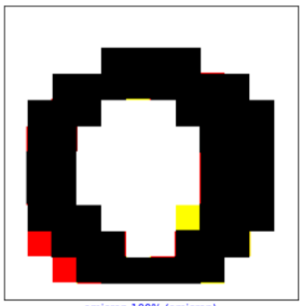
delta 27% represents the predicted label and (chi) represents the true label.

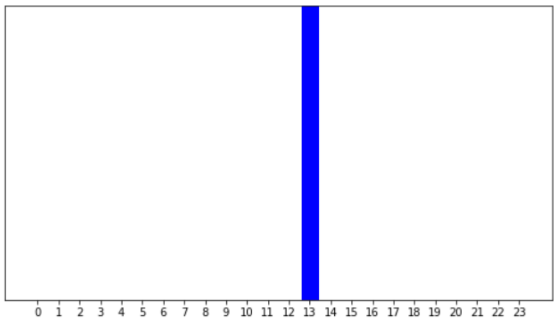
3.10.9 CNN trained model on High resolution Greek Characters with Low resolution Test dataset.

```
In [647]: predictions_2 = model_CNN_high.predict(x_test)

          import numpy as np
          i = 23
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions_2[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions_2[i], y_test)
          plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
predicted_label =  13
true_label in boolean list = [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, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0]
true_label after converting to integer = 13
```


omicron 100% (omicron)



As we have seen in the testing subsection "CNN appears to be the best when opposite datasets are used to train and test datasets". Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 23rd image in this case is "omicron" and as per the predict function and the CNN model omicron is predicted with 100% confidence as the predicted_label and the true label are equal to each other thus they are written in "blue" colour the bar plot shows 13 as the predicted label with highest confidence level which is predicted right.

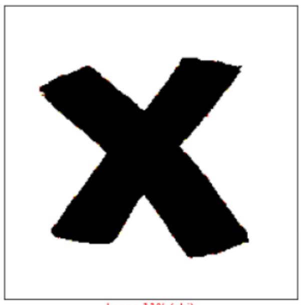
omicron 100% represents the predicted label and (omicron) represents the true label.

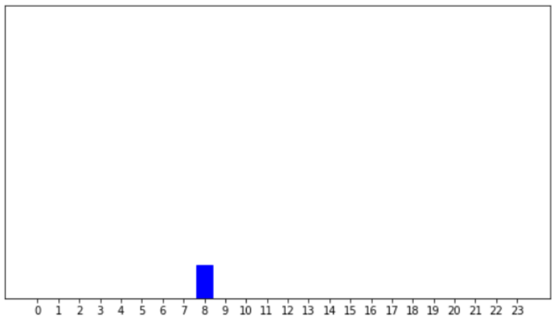
3.10.10 CNN trained model on Low resolution Greek Characters with High resolution Test dataset.

```
In [663]: predictions_5 = model_CNN.predict(x_test)

          import numpy as np
          i = 23
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions_5[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions_5[i], y_test)
          plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
predicted_label =  8
true_label in boolean list = [0.0, 0.0, 1.0, 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, 0.0, 0.0, 0.0]
true_label after converting to integer = 2
```

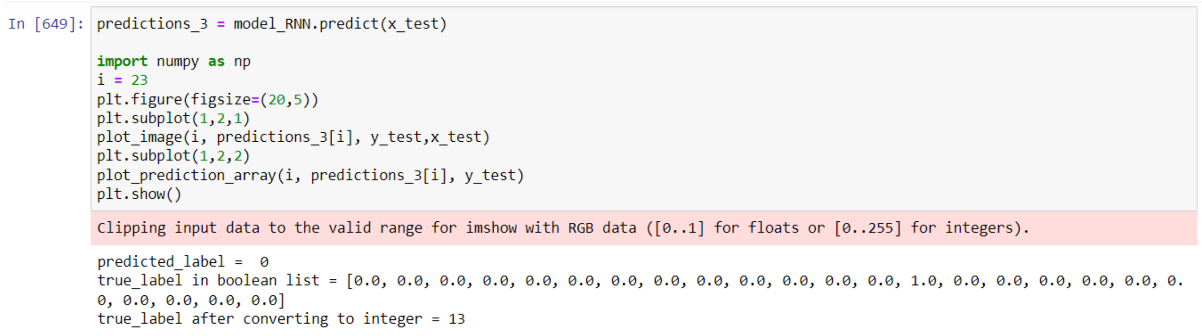

kappa 11% (chi)



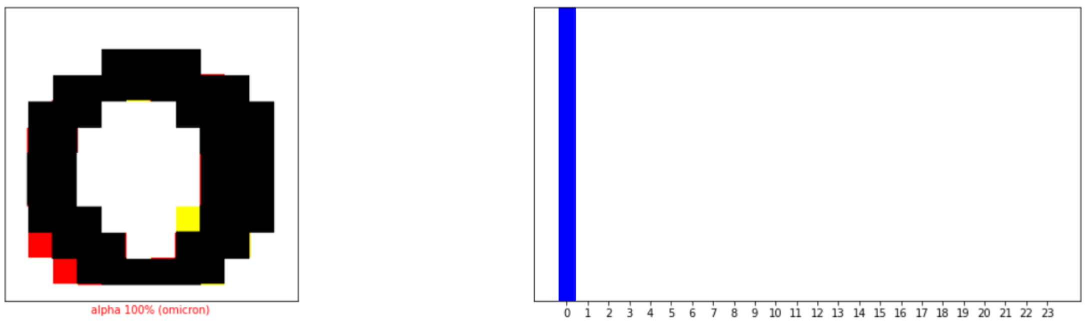
Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 23rd image in this case is "chi" and as per the predict function and the CNN model kappa is predicted with 11% confidence as the predicted_label and the true label are not equal to each other; they are written in "red" colour the bar plot shows 8 as the predicted label with highest confidence level which is predicted wrong as the true label of chi is 2.

kappa 11% represents the predicted label and (chi) represents the true label.

3.10.11 RNN trained model on High resolution Greek Characters with Low resolution Test dataset.



```
In [649]: predictions_3 = model_RNN.predict(x_test)

          import numpy as np
          i = 23
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions_3[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions_3[i], y_test)
          plt.show()

          Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

          predicted_label =  0
          true_label in boolean list = [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, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
          0, 0.0, 0.0, 0.0, 0.0]
          true_label after converting to integer = 13
```



Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 23rd image in this case is "omicron" and as per the predict function and the RNN model alpha is predicted with 100% confidence as the predicted_label and the true label are not equal to each other; they are written in "red" colour the bar plot shows 0 as the predicted label with highest confidence level which is predicted wrong as the true label of omicron is 13.

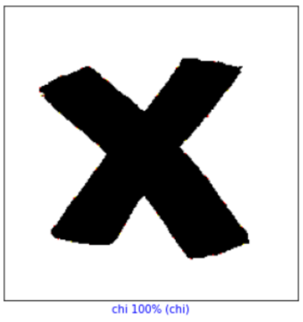
alpha 100% represents the predicted label and (omicron) represents the true label.

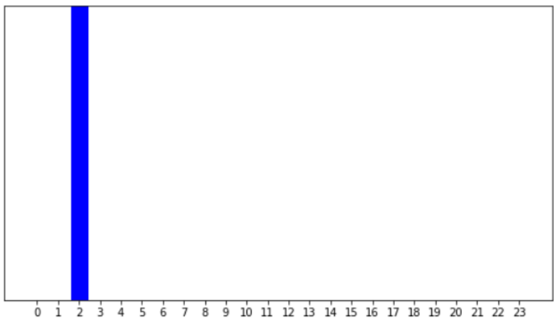
3.10.12 RNN trained model on Low resolution Greek Characters with High resolution Test dataset.

```
In [665]: predictions_6 = model_RNN_low.predict(x_test)

          import numpy as np
          i = 23
          plt.figure(figsize=(20,5))
          plt.subplot(1,2,1)
          plot_image(i, predictions_6[i], y_test,x_test)
          plt.subplot(1,2,2)
          plot_prediction_array(i, predictions_6[i], y_test)
          plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
predicted_label =  2
true_label in boolean list = [0.0, 0.0, 1.0, 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, 0.0, 0.0, 0.0]
true_label after converting to integer = 2
```





As we have seen in the testing subsection "RNN appears to be the better than ResNet50 when opposite datasets are used to train and test datasets". Here we have initialized "i" which we discussed earlier is the number of the image we want to predict. We want to predict the 23rd image in this case is "chi" and as per the predict function and the RNN model chi is predicted with 100% confidence as the predicted_label and the true label are equal to each other thus they are written in "blue" colour the bar plot shows 2 as the predicted label with highest confidence level which is predicted right.

chi 100% represents the predicted label and (chi) represents the true label.

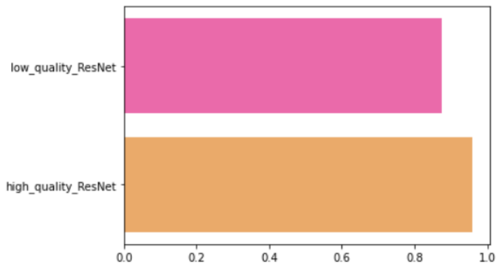
3.11 Comparing Accuracy

3.11.1 Accuracy for Low and High Quality Data (ResNet50)

```
In [782]: low = low_test_acc[0]
          high = test_acc[0]
          bar = [low,high]
```

```
In [783]: import seaborn as sns
          sns.barplot(x = bar,y = ["low_quality_ResNet","high_quality_ResNet"],palette = "spring")
```

```
Out[783]: <AxesSubplot:>
```



low_test_acc[0] and test_acc[0] is used to specify the first element from the list which represents ResNet50 model.

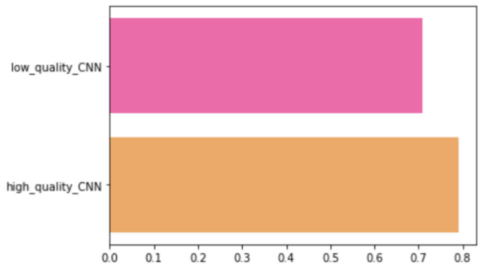
The above bar plot helps us visualize the difference in accuracy for ResNet50 model trained on low resolution Greek character dataset and high resolution Greek character dataset. Pink colour represents the accuracy for the ResNet50 model trained and tested on the low resolution Greek character dataset and the orange colour represents the accuracy for the ResNet50 model trained and tested on the high resolution Greek character dataset.

This visualization shows that ResNet50 model trained on the High resolution dataset has more accuracy than the ResNet50 model trained on the Low resolution dataset.

3.11.2 Accuracy for Low and High Quality Data (CNN)

```
In [784]: low = low_test_acc[1]
          high = test_acc[1]
          bar = [low,high]
          sns.barplot(x = bar,y = ["low_quality_CNN","high_quality_CNN"],palette = "spring")
```

```
Out[784]: <AxesSubplot:>
```



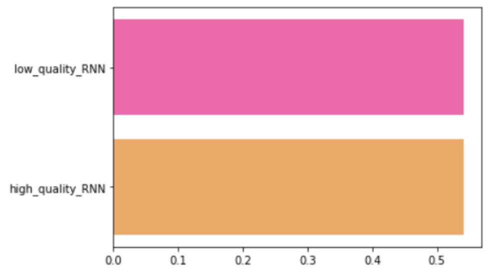
low_test_acc[1] and test_acc[1] is used to specify the second element from the list which represents CNN model.

The above bar plot helps us visualize the difference in accuracy for CNN model trained on low resolution Greek character dataset and high resolution Greek character dataset. Pink colour represents the accuracy for the CNN model trained and tested on the low resolution Greek character dataset and the orange colour represents the accuracy for the CNN model trained and tested on the high resolution Greek character dataset.

This visualization shows that CNN model trained on the High resolution dataset has more accuracy than the CNN model trained on the Low resolution dataset.

3.11.3 Accuracy for Low and High Quality Data (RNN)

```
In [687]: low = low_test_acc[2]
          high = test_acc[2]
          bar = [low,high]
          sns.barplot(x = bar,y = ["low_quality_RNN","high_quality_RNN"],palette = "spring")

Out[687]: <AxesSubplot:>
```



low_test_acc[2] and test_acc[2] is used to specify the third element from the list which represents RNN model.

The above bar plot helps us visualize the difference in accuracy for RNN model trained on low resolution Greek character dataset and high resolution Greek character dataset. Pink colour represents the accuracy for the RNN model trained and tested on the low resolution Greek character dataset and the orange colour represents the accuracy for the RNN model trained and tested on the high resolution Greek character dataset.

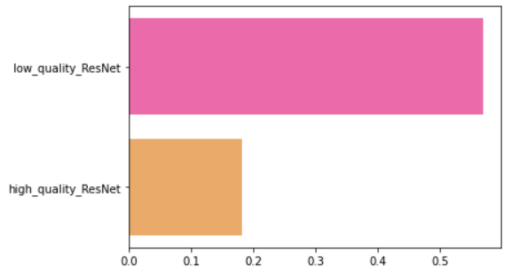
This visualization shows that RNN model trained on the High resolution dataset has the same accuracy as the RNN model trained on the Low resolution dataset.

3.12 Comparing Loss

3.12.1 Loss for Low and High Quality Data (ResNet50)

```
In [688]: low = low_test_error[0]
          high = test_error[0]
          bar = [low,high]
          sns.barplot(x = bar,y = ["low_quality_ResNet","high_quality_ResNet"],palette = "spring")
```

```
Out[688]: <AxesSubplot:>
```



low_test_error[0] and test_error[0] is used to specify the first element from the list which represents ResNet50 model.

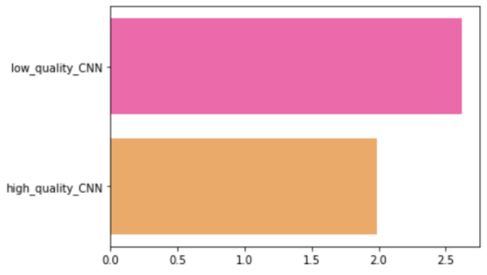
The above bar plot helps us visualize the difference in error or loss for ResNet50 model trained on low resolution Greek character dataset and high resolution Greek character dataset. Pink colour represents the error or loss for the ResNet50 model trained and tested on the low resolution Greek character dataset and the orange colour represents the error or loss for the ResNet50 model trained and tested on the high resolution Greek character dataset.

This visualization shows that ResNet50 model trained on the High resolution dataset has less error than the ResNet50 model trained on the Low resolution dataset.

3.12.2 Loss for Low and High Quality Data (CNN)

```
In [689]: low = low_test_error[1]
          high = test_error[1]
          bar = [low,high]
          sns.barplot(x = bar,y = ["low_quality_CNN","high_quality_CNN"],palette = "spring")
```

```
Out[689]: <AxesSubplot:>
```



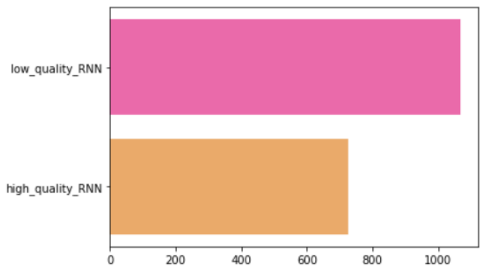
low_test_error[1] and test_error[1] is used to specify the first element from the list which represents CNN model.

The above bar plot helps us visualize the difference in error or loss for CNN model trained on low resolution Greek character dataset and high resolution Greek character dataset. Pink colour represents the error or loss for the CNN model trained and tested on the low resolution Greek character dataset and the orange colour represents the error or loss for the CNN model trained and tested on the high resolution Greek character dataset.

This visualization shows that CNN model trained on the High resolution dataset has less error than the CNN model trained on the Low resolution dataset.

3.12.3 Loss for Low and High Quality Data (RNN)

```
In [690]: low = low_test_error[2]
          high = test_error[2]
          bar = [low,high]
          sns.barplot(x = bar,y = ["low_quality_RNN","high_quality_RNN"],palette = "spring")

Out[690]: <AxesSubplot:>
```



low_test_error[2] and test_error[2] is used to specify the first element from the list which represents RNN model.

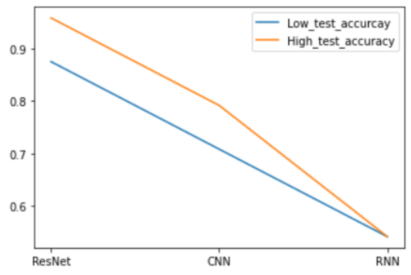
The above bar plot helps us visualize the difference in error or loss for RNN model trained on low resolution Greek character dataset and high resolution Greek character dataset. Pink colour represents the error or loss for the RNN model trained and tested on the low resolution Greek character dataset and the orange colour represents the error or loss for the RNN model trained and tested on the high resolution Greek character dataset.

This visualization shows that RNN model trained on the High resolution dataset has less error than the RNN model trained on the Low resolution dataset. But compared to the other models it has the most error percentage.

4. Conclusion

4.1 Comparing Accuracy of all the Models.

```
In [791]: plt.plot(["ResNet","CNN","RNN"],low_test_acc)
          plt.plot(["ResNet","CNN","RNN"],test_acc)
          plt.legend(['Low_test_accurcay',"High_test_accuracy"])

Out[791]: <matplotlib.legend.Legend at 0x1b2169476d0>
```



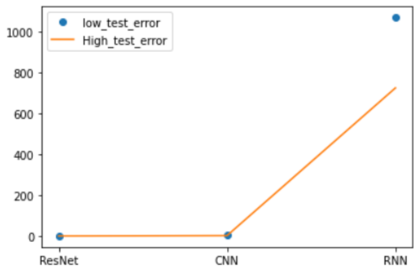
The line plot shown above helps us understand the accuracy we have got throughout all the models and all the high resolution and low resolution dataset.

Blue colour represents the accuracy of the models trained on low resolution Greek character dataset and the orange colour represents the accuracy of the models trained on high resolution Greek character dataset.

The line plot clearly states that accuracy represented by orange and blue line descends as it moves from ResNet50 model which has the best accuracy percentage to CNN which has a moderate accuracy percentage to RNN model which has the least accuracy percentage.

4.2 Comparing Error of all the Models.

```
In [790]: plt.plot(["ResNet","CNN","RNN"],low_test_error,"o")
          plt.plot(["ResNet","CNN","RNN"],test_error)
          plt.legend(['low_test_error',"High_test_error"])

Out[790]: <matplotlib.legend.Legend at 0x1b342dbfc10>
```



The line and dot plot shown above helps us understand the Error we have got throughout all the models and all the high resolution and low resolution dataset.

Blue dots represents the Error of the models trained on low resolution Greek character dataset and the orange line represents the Error of the models trained on high resolution Greek character dataset.

The line and dot plot clearly states that error represented by orange and blue line ascends as it moves from ResNet50 model which has the least error percentage to CNN which has a moderate error percentage to RNN model which has the most error percentage.

4.3 The Final comparison

In a nutshell when ResNet50, CNN and RNN models were trained on High Resolution Greek character dataset as well as on Low Resolution Greek character dataset The accuracy was always high for ResNet50 model whereas the accuracy was always low for the RNN model, the error percentage for ResNet50 model was always low for both the datasets but RNN had the least performance and highest error percentage for both the datasets. The CNN model gave good and moderate results in accuracy and error for both the datasets when compared to ResNet50 and when compared to RNN it gave a lot better result.

When the Models were trained on a different dataset (e.g. High resolution Greek character dataset) and then tested on a different dataset (e.g. Low resolution Greek character dataset), I observed some interesting results. In this scenario CNN proved to be the best model followed by RNN with moderate results and in the end ResNet50 with very less accuracy.

5. How to run the code.

The python code is pretty easy to work with and is not so complicated however *Jupyter notebook* should be pre-installed as the code submitted is a jupyter notebook (.IPYNB) format. Two training datasets are provided one with High resolution Greek dataset named Input_dataset and the other one with Low resolution Greek dataset Input_dataset_low, same goes for the 2 test datasets for high and low resolution datasets named test_high and test_low respectively.

If any of the modules gives an import error simply type pip install "name of the module not installed" in the command prompt and this will help you with installing the required module, and also help you get rid of the error.

The MyProject zip folder contains following items:

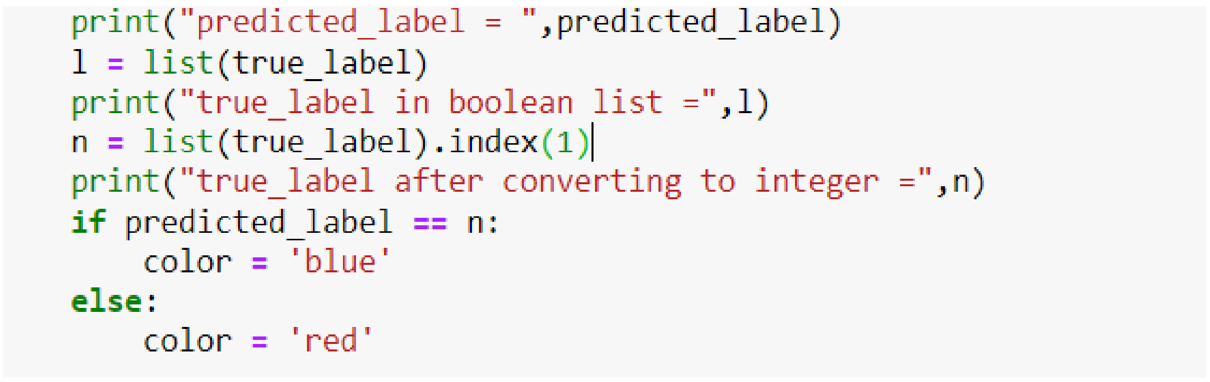
1. Input_dataset
2. Input_dataset_low
3. processed_data
4. processed_data_low
5. test_high
6. test_low
7. project_report
8. Project_Final_Greek_letters_recognition-final.IPYNB (Jupyter Notebook)

6. Problems encountered during the Project building phase.

The first problem I faced was during the background research phase. As the dataset for Greek characters is rare to find, finding the right dataset and understanding the characters was one challenge but I enjoyed the process of understanding the dataset I worked on.

The second problem I encountered was during the experiment phase. Understanding and working with tensorflow and keras is bit tricky, but with extensive research, trial and errors I got a hang of it. Another problem was to decide on how much percentage should I split the train and validation set, 60 – 40 was the final split percentage I chose to split my input data.

Working with a Boolean array was challenging to work with as true_label was in the format of Boolean array and to test whether the accuracy we got on the model is correct and till what percentage (confidence), we need to compare true_label (Boolean array) to the predicted label (Integer) which is not possible as it raises an error.



```
print("predicted_label = ",predicted_label)
l = list(true_label)
print("true_label in boolean list =",l)
n = list(true_label).index(1)
print("true_label after converting to integer =",n)
if predicted_label == n:
    color = 'blue'
else:
    color = 'red'
```

To overcome the above issue, I converted the array into a list using the list() function and then to convert the list into an integer I have simply used .index(1) function to count the number of the index which has 1 as its value in the list. Here n is the variable used to save the number of the index which is further used in the if statement to compare if n (true label in integer format) equal to the predicted label.

7. Future additions to the Project.

In the future I aim to use a polytonic Greek character dataset as it is unavailable currently but, in the future, I aim to make my own dataset and feed my current software with that dataset.

I also aim to add more models to compare the accuracy.

Further this software can be used in the future with a different language dataset with some changes done according to the dataset I aim to use.

Report References

Greek Alphabet | The Greek Alphabet, Greek Letter, Greek Alphabets, Greek Characters | Greece.com. Available at: http://www.greece.com/info/language/greek_alphabet/ (Accessed: Nov 11, 2021).

Baheti, P. (2021) *12 Types of Neural Networks Activation Functions: How to Choose?* Available at: https://www.v7labs.com/blog/neural-networks-activation-functions, https://www.v7labs.com/blog/neural-networks-activation-functions (Accessed: Nov 13, 2021).

Balaji, S. (2020) *Binary Image classifier CNN using TensorFlow.* Available at: https://medium.com/techiepedia/binary-image-classifier-cnn-using-tensorflow-a3f5d6746697 (Accessed: Nov 30, 2021).

Boesch, G. (2021) 'Deep Residual Networks (ResNet, ResNet50) - Guide in 2021', *viso.ai,* -08-29T14:26:41+00:00. Available at: https://viso.ai/deep-learning/resnet-residual-neural-network/ (Accessed: Nov 13, 2021).

Grandperrin, J. (2021) 'How to use confidence scores in machine learning models', *Medium,* -02-01T15:56:25.964Z. Available at: https://towardsdatascience.com/how-to-use-confidence-scores-in-machine-learning-models-abe9773306fa (Accessed: Nov 13, 2021).

He, K., Zhang, X., Ren, S. and Sun, J. (2015) 'Deep Residual Learning for Image Recognition', .

HMLDude SCBoy (2018) *optimization - Difference Between Rho and Decay Arguments in Keras RMSprop.* Available at: https://stats.stackexchange.com/questions/351409/difference-between-rho-and-decay-arguments-in-keras-rmsprop (Accessed: Nov 10, 2021).

https://www.loc.gov/catdir/cpso/romanization/greek.pdf (2010) *Greek (ALA-LC Romanization Tables).* Library of Congress. Available at: https://www.loc.gov/catdir/cpso/romanization/greek.pdf (Accessed: Nov 09, 2021).

JACKOWSKI, M. and JACKOWSKI, P. (2021) 'Image Recognition in Python based on Machine Learning – Example & Explanation for Image Classification Model | ASPER BROTHERS', *asperbrothers.com,* -07-30T10:42:03+00:00. Available at: https://asperbrothers.com/blog/image-recognition-in-python/ (Accessed: Nov 12, 2021).

Khandelwal, R. (2019) 'Deep Learning using Transfer Learning -Python Code for ResNet50', *Medium,* -11-27T15:34:14.929Z. Available at: https://towardsdatascience.com/deep-learning-using-transfer-learning-python-code-for-resnet50-8acdfb3a2d38 (Accessed: Nov 13, 2021).

Learning, G. (2020) *How To Apply Machine Learning to Recognise Handwriting.* Available at: https://www.mygreatlearning.com/blog/how-to-recognise-handwriting-with-machine-learning/ (Accessed: Nov 12, 2021).

Mahmood, A., Giraldo, A., Bennamoun, M., An, S., Sohel, F., Boussaid, F., Hovey, R., Fisher, R. and Kendrick, G. (2020) 'Automatic Hierarchical Classification of Kelps Using Deep Residual Features.', *Sensors,* 20, pp. 447. doi: 10.3390/s20020447.

Rhul.ac.uk (2021) *2021 Artificial Intelligence Individual Projects List.* Available at: http://projects.cs.rhul.ac.uk/List2021Jan.php?PROJECT-TYPE=AI (Accessed: Apr 07, 2021).

Saha, S. (2018) 'A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way', *Medium,* -12-17T05:02:27.450Z. Available at: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (Accessed: Nov 30, 2021).

Sahoo, S. (2021) 'Residual blocks — Building blocks of ResNet', *Medium,* -08-16T04:27:01.346Z. Available at: https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec (Accessed: Nov 30, 2021).

Saxena, S. (2021) 'Image Augmentation Techniques for Training Deep Learning Models', *analyticsvidhya.com,* Mar 10,. Available at: https://www.analyticsvidhya.com/blog/2021/03/image-augmentation-techniques-for-training-deep-learning-models/ (Accessed: .

Team, K. (2020) *Keras documentation: Transfer learning & fine-tuning.* Available at: https://keras.io/guides/transfer_learning/ (Accessed: Nov 13, 2021).

Wood, T. (2020) *Sigmoid Function.* Available at: https://deepai.org/machine-learning-glossary-and-terms/sigmoid-function (Accessed: Nov 28, 2021).

Wood, T. (2019) *Softmax Function.* Available at: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer (Accessed: Nov 28, 2021).

Dataset Reference

Kontolati, K. (2020) 'Automatic Hierarchical Classification of Kelps Using Deep Residual Features', *Sensors,* January 13,. Available at: https://www.kaggle.com/katianakontolati/classification-of-handwritten-greek-letters (Accessed: Nov 01, 2021).

Code References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu and Xiaoqiang Zheng (2015) 'TensorFlow: Large-scale machine learning on heterogeneous systems', .

Team, K. (2020) *Keras documentation: The Sequential model.* Available at: https://keras.io/guides/sequential_model/ (Accessed: Nov 14, 2021).

TensorFlow (2021) *Basic classification: Classify images of clothing.* Available at: https://www.tensorflow.org/tutorials/keras/classification (Accessed: Nov 20, 2021).

Train Neural Network by loading your images |TensorFlow, CNN, Keras tutorial (2020) Directed by Bhatt, J. [Youtube.com].

Training Residual Neural Network with your own dataset (2020) Directed by Bhatt, J. [Youtube.com].

Recurrent Neural Networks (RNN) - Deep Learning w/ Python, TensorFlow & Keras p.7 (2018) Directed by Kinsley, H. [Youtube.com].

Matplotlib (2012) 'Plot types | Overview of many common plotting commands in Matplotlib.', .

Mohdsanadzakirizvi. (2020) 'CNN Image Classification | Image Classification Using CNN', *Analytics Vidhya,* -02-18T02:45:31+00:00. Available at: https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/ (Accessed: Nov 01, 2021).

PyTorch (2017) 'TORCHVISION', .

Python (2020) 'split-folders 0.4.3', .

Python (2021) 'os 3.10.1', .

Python (2021) 'Numpy 1.21.0', .

Python (2021) 'pandas 1.3.4', .

Python (2021) 'scikit-learn 1.0.0', .

Python (2021) ‘opencv-python 4.5.4.60’, .

Waskom, M. (2020) *Seaborn | User guide and tutorial.* Available at: https://seaborn.pydata.org/tutorial.html#user-guide-and-tutorial (Accessed: Apr 07, 2021).

xiaochus and Sentdex 'Python keras.layers.GlobalAveragePooling2D() Examples', .