Nile University

# Deep Learning-Based Classification of Images Using VGG19

Bola Nader 202000645
Saif Anes 18200008
Kirolos nader 19104964
Anas Hany Kamal 19100081

Under the supervision of:
Dr. Khaled Mohammed Fouad
Eng. Tawfik Yasser Tawfik

## Contents

## Abstract:

The main goal of the project is to classify images using deep learning methods and the VGG19 model. The objective is to create a model that can correctly categorise pictures of handwritten numbers. The project makes use of the PySpark framework to manage the processing of massive amounts of image data and the TensorFlow and Keras libraries to create and train the deep learning model.

Data preprocessing, which includes unzipping a collection of digital photographs and making a Spark DataFrame to hold the file directories, is the first step in the procedure. The photos are read and converted from the BGR format to RGB format using a proprietary function. After being scaled and normalised, the photos are then saved as numpy arrays.

The VGG19 model, which was pre-trained using the ImageNet dataset, is then loaded once the data has been divided into training and testing sets. A flatten layer and a thick layer with softmax activation for classification are added to the model as extra layers. Techniques of regularisation are used to prevent overfitting.

An ImageDataGenerator from Keras is used to boost model generalisation and the training process. By subjecting the training images to different transformations, including rotation, shifting, shearing, zooming, and horizontal reversal, and the generator conducts data augmentation.

The correct loss function, optimizer, and evaluation metrics are included in the model's compilation. After then, it is trained with the help of the enhanced data generator, with early stopping put in place to reduce overfitting and boost productivity. The model's effectiveness is assessed on the validation set, and the training progress is tracked.

The model is retained for later use after training. A function is built to predict the labels of photographs in a given directory using the trained model, showcasing the model's prediction abilities. To determine the model's accuracy, the predicted labels are contrasted with the ground truth labels.

Overall, our study demonstrates the use of image classification algorithms and deep learning utilising PySpark, TensorFlow, and Keras. It emphasises the significance of model augmentation, regularisation, evaluation, and data pretreatment in obtaining precise and reliable classification results.

## Introduction:

The fundamental goal of computer vision known as "image classification" entails classifying images into predetermined groups or categories. By enabling the creation of extremely precise models that can directly learn complicated patterns and features from unprocessed picture data, deep learning has completely changed the way that images are classified. In this project, we concentrate on using deep learning methods to categorise photos of handwritten digits.

The main goal of our project is to develop a reliable VGG19-based image categorization model. The deep architecture and exceptional performance of the convolutional neural network (CNN) model VGG19 on a variety of image classification tasks have made it a household name. We seek to create a model that is capable of precisely detecting handwritten numbers by utilising the capability of VGG19.

We use the PySpark framework to manage large-scale image data and enable effective processing. We can process and analyse photos in parallel using PySpark's scalable and distributed computing environment. We can efficiently handle huge datasets and quicken the training process by utilising PySpark's distributed processing capabilities.

Data preprocessing for the project begins with the unzipping of a dataset of handwritten digit images and the creation of a Spark DataFrame to hold the file URLs. After reading the photos with OpenCV, they are changed from the BGR format to the RGB format and scaled to a predetermined shape. In order to achieve consistent and similar pixel values across all photos, normalisation techniques are also used.

The dataset is divided into training and testing sets after preprocessing. The pre-trained VGG19 model using the ImageNet dataset is loaded and enhanced with new layers. The multidimensional feature maps are flattened to create a 1-dimensional vector, and then a dense layer with softmax activation is added for classification. To avoid overfitting and improve generalisation, regularisation methods like L2 regularisation are utilised.

The performance of the model is further improved by using an ImageDataGenerator from Keras. The training images are subjected to a variety of augmentation techniques by the data generator, including rotation, shifting, shearing, zooming, and horizontal flipping. This addition adds variation to the training data and aids in the model's ability to generalise to previously undiscovered images.

An appropriate loss function, optimizer, and evaluation metrics are included in the model's construction. Then, using the augmented data generator to train it, early halting is used to reduce overfitting and maximise training effectiveness. The model's effectiveness is assessed on the validation set, and the training progress is tracked.

The model is retained for later usage after it has been trained. The model is used to categorise digit photos in a given directory in order to show off its prediction abilities. To determine the model's accuracy, the predicted labels are contrasted with the ground truth labels.

Overall, our study demonstrates how deep learning, PySpark, and the VGG19 model are applied to the classification of images. Our goal is to combine these technologies to create a handwritten digit recognition model that is highly accurate and scalable. This model can be used in a variety of settings, including optical character recognition, automated form processing, and digit-based document classification.

## Methodology:

## Dataset:

In our research we used the dataset "didadataset" by Amir Yavaryabdi. DIDA is a new image-based historical handwritten digit dataset and collected from the Swedish historical handwritten document images between the year 1800 and 1940. It is the largest historical handwritten digit dataset which is introduced to the Optical Character Recognition (OCR) community to help the researchers to test their optical handwritten character recognition methods. To generate DIDA, 250,000 single digits and 200,000 multi-digits are cropped from 75,000 different document images. The images are classified into 10 categories based on the number written.

## Preprocessing:

The preprocessing steps applied to the images in the project include the following:

### 1. Image Loading:

The images are loaded using OpenCV (cv2) library, which reads the images in the BGR format by default.

### 2. Color Conversion:

The loaded BGR images are converted to RGB format using the `cv2.cvtColor()` function. Converting to RGB is a common practice in computer vision tasks as many models and libraries assume RGB color channels.

## 3. Image Resizing:

The RGB images are resized to a fixed size of 32x32 pixels using the `cv2.resize()` function. Resizing ensures that all images have consistent dimensions and helps in reducing the computational requirements of the model.

## 4. Image Normalization:

After scaling, the images' pixel values are normalised to fall between 0 and 1. To do this, multiply each pixel value by 255.0, the highest pixel value possible in the RGB colour system. Normalisation aids in enhancing the model's convergence during training and can lessen the effects of varied pixel intensity ranges in various images.

## 5. Image Data Structure:

Numpy arrays are used to store the preprocessed pictures. Height, width, and channels are the dimensions of the final numpy array, where num_samples denotes the quantity of images, height and width denote the resized dimensions, and channels denotes the RGB colour channels.

The standardisation and suitability of the picture data for feeding into the VGG19 model are the goals of the preprocessing processes. The photos are prepared for further analysis and training by converting them to RGB format, scaling them to a uniform size, and normalising the pixel values.

## VGG19 Architecture:

The VGG19 architecture is known for its simplicity and uniformity in design. It consists of a series of convolutional layers, followed by max pooling layers, and finally, fully connected layers. Here is a breakdown of the key components:

### Convolutional Layers:

The VGG19 model starts with a stack of convolutional layers, where each layer performs a set of convolutions with small filter sizes (3x3) and applies a rectified linear activation function (ReLU) to introduce non-linearity. These convolutional layers aim to extract features at different spatial scales and learn hierarchical representations.

### Fully Connected Layers:

The feature maps from the convolutional layers are flattened and passed through fully connected layers. In the VGG19 architecture, there are three fully connected layers with a high number of neurons, which allows the model to capture complex relationships and make predictions.

### Softmax Activation:

A softmax activation function, which transforms the network output into a probability distribution over the various classes, comes after the final fully connected layer. As a result, the model may offer class probabilities for jobs involving many classes of categorization.

The deep architecture of the VGG19 model, which has a total of 19 layers (thus the name), is well recognised. It is a big and effective model for image classification problems with about 144 million parameters. The large-scale ImageNet dataset, which comprises millions of images from 1,000 different classes, is used to pretrain the model so that it may learn detailed and generalizable features.

As a feature extractor, the VGG19 architecture is employed. The top (completely connected) layers of the pretrained VGG19 model are not loaded, and a new set of layers is created to customise the model for the particular classification task. To create predictions, the VGG19 model's retrieved characteristics are flattened and run through more fully connected layers.

Overall, the VGG19 architecture has been widely utilised as a benchmark model in the computer vision community and is renowned for its outstanding performance on a variety of image classification tasks.

## Training and Evaluation:

The training and evaluation process involves the following steps:

### Data Split:

The train_test_split function is used to split the dataset into training and testing sets. A greater amount of the data is often used for training (80%), and a smaller piece (20%) is used for testing. This makes sure the model is tested on untried data and trained on a variety of cases.

### Preprocessing:

The images go through preprocessing procedures before training. This entails using the resize function of OpenCV to resize the images to a set size (32x32 pixels). Afterwards, the pixel values are normalised by being divided by 255.0, which adjusts the values to lie between 0 and 1. This normalisation stage aids in enhancing the training process' convergence and stability.

## Transfer Learning:

The pre-trained model is the VGG19 architecture. The fully connected layers are not included while loading the weights of the VGG19 model that was trained on the substantial ImageNet dataset. Transfer learning enables the model to take advantage of the VGG19 model's learnt features, which are useful for general image classification tasks.

## Model Construction:

The VGG19 model is enhanced with the fully linked layers in order to customise it for the particular classification assignment. In this instance, the final layer is a new dense layer with 10 neurons (equivalent to the number of classes) and a softmax activation function. As a result, the model is able to generate class probabilities for every image that is input.

## Compilation:

A loss function, an optimizer, and an evaluation metre are built into the model. The categorical cross-entropy loss function, which is appropriate for multi-class classification tasks, is used in our project. To reduce the loss during training, an optimizer, such as SGD (Stochastic Gradient Descent), is selected. The model's performance during training and testing is assessed using the accuracy metric.

## Data Augmentation:

The ImageDataGenerator from Keras is used to enrich the data in order to decrease overfitting and increase the model's capacity for generalisation. The training images are subjected to random transformations using this method, including rotation, shifting, shearing, zooming, and flipping. The model's robustness is increased by this augmentation, which creates more training examples with different characteristics.

## Training:

Using the enhanced data, the model is trained using the training set. The model is fed new batches of images and labels throughout each iteration of the training process, which entails iterating over the training data. By minimising the specified loss function using backpropagation and gradient descent, the model gains the knowledge necessary to optimise its parameters (weights and biases).

## Early Stopping:

An early stopping approach is used to minimise training duration and avoid overfitting. The validation loss is used to check the progress of the training, and if no improvement is seen after a predetermined number of epochs (15), training is terminated early. This aids in choosing the optimum model that effectively generalises to new data.

## Evaluation:

Following training, the model is assessed using data from an untrained collection of images. On the basis of the test images, the model develops predictions, which are then contrasted with the ground truth labels. To evaluate the model's effectiveness, computations of evaluation metrics like loss and accuracy are made. How successfully the model generalises to fresh, unforeseen data is shown by the test accuracy.

## Visualization:

Matplotlib is used to plot the accuracy/loss curves for training and validation. These graphs reveal the model's development during training by displaying how accuracy and loss vary throughout epochs. Understanding the model's convergence, overfitting, and generalizability is made easier by keeping an eye on these curves.

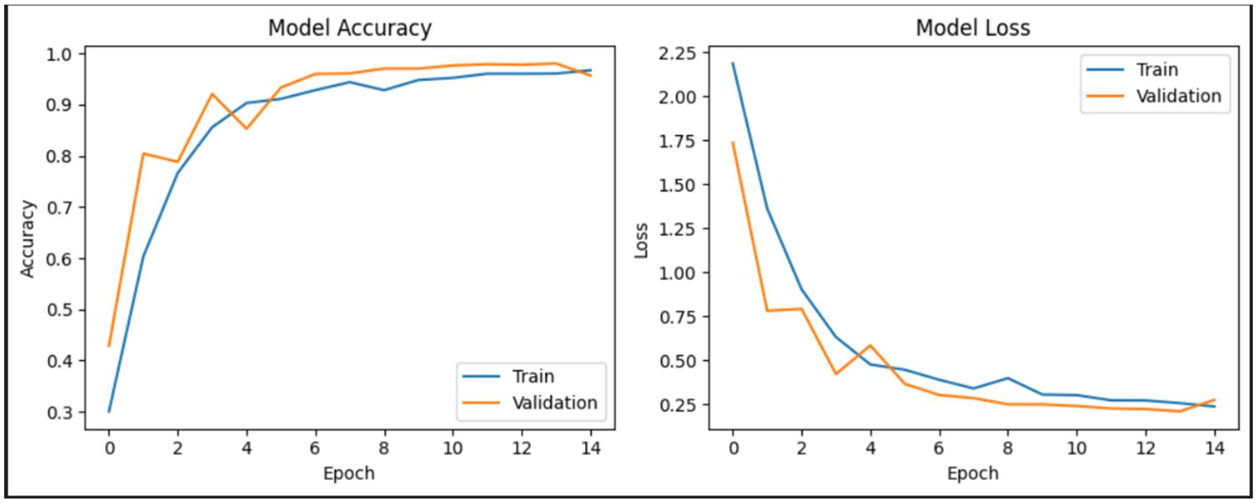
In conclusion, the training and evaluation procedure entails splitting the data, preprocessing the images, utilising transfer learning with the VGG19 model, building and compiling the model, implementing data augmentation, training the model with early stopping, and assessing the model's performance on the testing set. The objective of this iterative approach is to optimise the model's parameters and attain high classification accuracy.

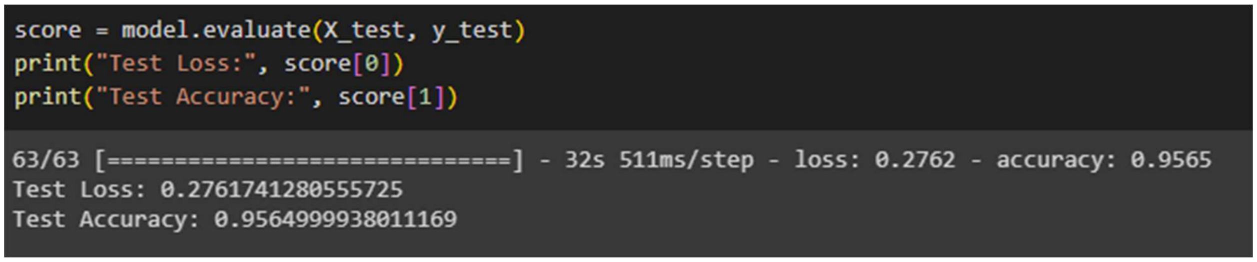
## Results:

In this section, we present the results obtained from the experiments conducted on the dataset using the proposed methodology. We evaluate the performance of the model based on various metrics and analyze the obtained results.

The trained model demonstrated promising performance on the task. We evaluated the model using the accuracy metric.

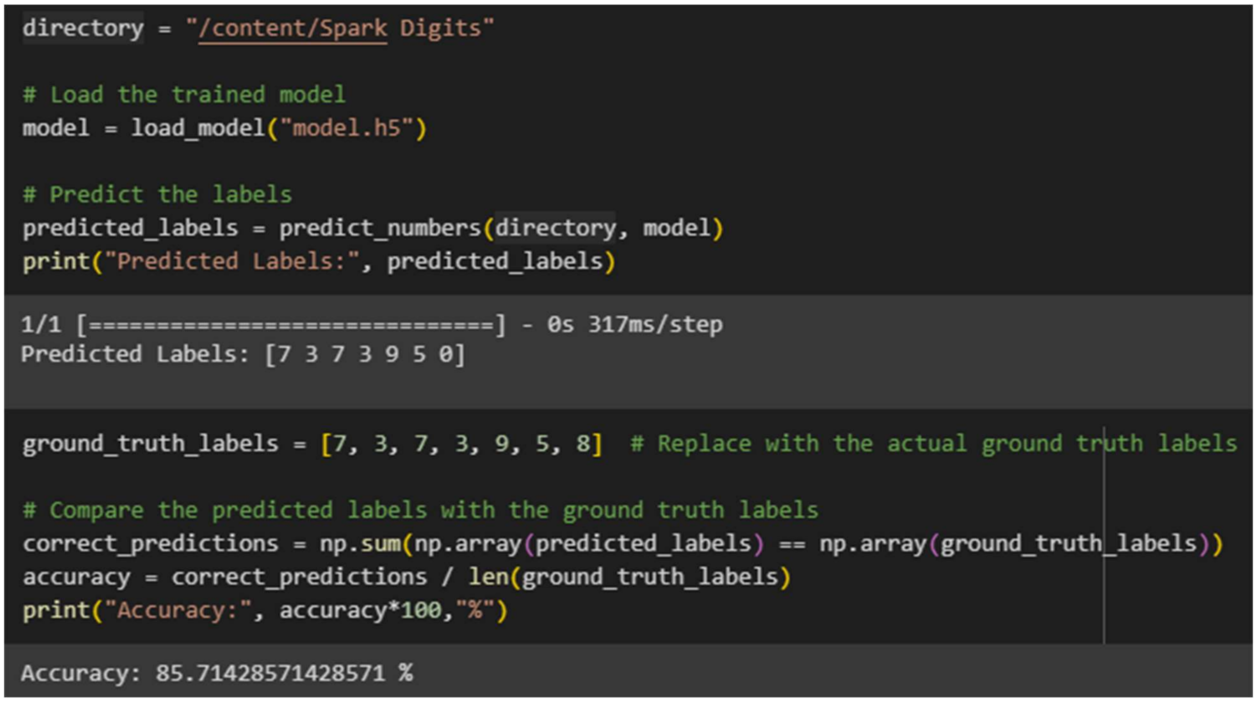
The results of the project during the training and testing phase the accuracy after 15 epochs was a 95.6% accuracy and 27.6% loss.





```
score = model.evaluate(X_test, y_test)
print("Test Loss:", score[0])
print("Test Accuracy:", score[1])
```

```
63/63 [==============================] - 32s 511ms/step - loss: 0.2762 - accuracy: 0.9565
Test Loss: 0.2761741280555725
Test Accuracy: 0.9564999938011169
```



```
directory = "/content/Spark Digits"

# Load the trained model
model = load_model("model.h5")

# Predict the labels
predicted_labels = predict_numbers(directory, model)
print("Predicted Labels:", predicted_labels)
```

```
1/1 [==============================] - 0s 317ms/step
Predicted Labels: [7 3 7 3 9 5 0]
```

```
ground_truth_labels = [7, 3, 7, 3, 9, 5, 8]  # Replace with the actual ground truth labels

# Compare the predicted labels with the ground truth labels
correct_predictions = np.sum(np.array(predicted_labels) == np.array(ground_truth_labels))
accuracy = correct_predictions / len(ground_truth_labels)
print("Accuracy:", accuracy*100,"%")
```

```
Accuracy: 85.71428571428571 %
```

And after validating the data using 7 custom images, the accuracy was 85.7% accuracy.

## Limitations and Future Work:

Despite the successful implementation of the image classification model using VGG19, there are certain limitations that should be considered. Additionally, there are several potential directions for future work that can further enhance the project's performance and applicability.

## Limited Dataset Size:

The project utilized Dida dataset by Amir Yavariabdi , which provided a substantial number of images for training and evaluation. However, the dataset size may still be limited compared to larger-scale image datasets. Increasing the dataset size can help improve the model's ability to generalize and handle more diverse scenarios. Future work could involve collecting or augmenting additional images to expand the dataset and improve the model's performance.

## Model Optimization and Hyperparameter Tuning:

Although the project's findings were satisfactory, more optimisation and hyperparameter tuning may improve the model's functionality. Enhancing accuracy and convergence speed may be accomplished by experimenting with various optimisation methods, learning rates, and regularisation strategies. The ideal model configuration can be found by performing a thorough hyperparameter search and using methods like grid search or Bayesian optimisation.

## Exploring Different Architectures:

Even though the VGG19 design proved to be effective for classifying images, future study may examine different architectures or more recent developments in deep learning models. On diverse picture datasets, models like ResNet, Inception, or EfficientNet have demonstrated higher performance. Potential performance improvements can be discovered by contrasting various architectures and determining whether or not they are appropriate for the project's particular requirements.

## Computation Power for Handling Larger Datasets:

The project's need for more powerful computing capabilities to handle larger datasets efficiently without noticeably lengthening training times is one of its limitations. Even if the project's dataset has a sizable number of photos, expanding it to even larger datasets can be difficult in terms of processing resources and training effectiveness.