**Group 18:**

**Nguyễn Thiên Bảo – ITITWE19022**

**Digit recognition**

**Project Overview**

The goal of this project is to develop a neural network model that can accurately classify handwritten digits from the MNIST dataset. The MNIST dataset is a well-known dataset in the field of machine learning and computer vision, often used as a benchmark for new algorithms.

Models: EfficientNetV2, inception\_v3 ,resnet\_v2,MNISTRecog

**Key Components**

**1. Dataset**

* **MNIST Dataset**: This dataset contains 70,000 grayscale images of handwritten digits (0-9), each of size 28x28 pixels. There are 60,000 training images and 10,000 test images.
* The data files train.csv and test.csv contain gray-scale images of hand-drawn digits, from zero through nine.
* Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.
* The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.
* Each pixel column in the training set has a name like pixelx, where x is an integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed x as x = i \* 28 + j, where i and j are integers between 0 and 27, inclusive. Then pixelx is located on row i and column j of a 28 x 28 matrix, (indexing by zero).

**2. Libraries Used**

* **numpy**: For numerical operations.
* **pandas**: For data manipulation (though not heavily used in this specific project).
* **keras**: For building and training the neural network.
* **matplotlib**: For visualizing the images and the results.

**3. Model Architecture**

* **Input Layer**: The input layer takes in the 28x28 pixel images.
* **Convolutional Layers**: Two convolutional layers with ReLU activation functions for feature extraction.
* **Pooling Layer**: A MaxPooling layer to reduce the spatial dimensions.
* **Dropout Layer**: A Dropout layer to prevent overfitting.
* **Flatten Layer**: Flattens the 3D output to 1D for the dense layers.
* **Dense Layers**: Fully connected layers for classification, including an output layer with 10 neurons (one for each digit) and a softmax activation function.

**4. Preprocessing**

* **Reshape**: Reshaping the data to fit the input shape expected by the convolutional layers.
* **Normalization**: Scaling the pixel values to the range [0, 1].
* **One-Hot Encoding**: Converting the labels to one-hot encoded vectors.

**5. Training**

* **Optimizer**: Adadelta optimizer is used.
* **Loss Function**: categorical\_crossentropy loss function is used for multi-class classification.
* **Metrics**: Accuracy is used to evaluate the performance of the model.
* **Epochs and Batch Size**: The model is trained for 150 epochs with a batch size of 500.

**6. Evaluation**

* The model's performance is evaluated on the test set, providing metrics like loss and accuracy.

**7. Visualization**

* **Sample Images**: Visualizing sample images from the training set.
* **Predictions**: Making predictions on the test set and visualizing the results.

**Code Summary**

**Import Libraries**

import numpy as np

import pandas as pd

import keras

from keras.datasets import mnist

from keras.models import Model

from keras.layers import Dense, Input, Conv2D, MaxPooling2D, Dropout, Flatten

from keras import backend as k

import matplotlib.pyplot as plt

**Load and Visualize Data**

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()for i in range(9):

plt.subplot(330 + 1 + i)

plt.imshow(x\_train[i], cmap=plt.get\_cmap('gray'))

plt.show()

**Preprocess Data**

img\_rows, img\_cols = 28, 28

if k.image\_data\_format() == 'channels\_first':

x\_train = x\_train.reshape(x\_train.shape[0], 1, img\_rows, img\_cols)

x\_test = x\_test.reshape(x\_test.shape[0], 1, img\_rows, img\_cols)

inpx = (1, img\_rows, img\_cols)

else:

x\_train = x\_train.reshape(x\_train.shape[0], img\_rows, img\_cols, 1)

x\_test = x\_test.reshape(x\_test.shape[0], img\_rows, img\_cols, 1)

inpx = (img\_rows, img\_cols, 1)

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

x\_train /= 255

x\_test /= 255

y\_train = keras.utils.to\_categorical(y\_train)

y\_test = keras.utils.to\_categorical(y\_test)

**Build and Compile Model**

inpx = Input(shape=inpx)

layer1 = Conv2D(32, kernel\_size=(3, 3), activation='relu')(inpx)

layer2 = Conv2D(64, (3, 3), activation='relu')(layer1)

layer3 = MaxPooling2D(pool\_size=(3, 3))(layer2)

layer4 = Dropout(0.5)(layer3)

layer5 = Flatten()(layer4)

layer6 = Dense(250, activation='sigmoid')(layer5)

layer7 = Dense(10, activation='softmax')(layer6)

model = Model([inpx], layer7)

model.compile(optimizer=keras.optimizers.Adadelta(),

loss=keras.losses.categorical\_crossentropy,

metrics=['accuracy'])

**Train Model**

model.fit(x\_train, y\_train, epochs=150, batch\_size=500)

**Evaluate Model**

score = model.evaluate(x\_test, y\_test, verbose=0)

print('loss=', score[0])

print('accuracy=', score[1])

**Make Predictions and Visualize**

predictions = model.predict(x\_test)

print(np.argmax(np.round(predictions[9])))

plt.imshow(x\_test[9].reshape(28, 28), cmap=plt.cm.binary)

plt.show()

**Conclusion**

This project demonstrates the process of building, training, and evaluating a simple neural network model using the MNIST dataset.

It covers data preprocessing, model architecture design, training, and evaluation, providing a solid foundation for understanding basic deep learning concepts and techniques.

**Reference**

This datasets [Digit Recognizer | Kaggle](https://www.kaggle.com/c/digit-recognizer/data)

**History runtime:**

<keras.src.callbacks.history.History at 0x15fffd74dd0>.history.History at 0x15fffd74dd0>