## **Artificial Intelligence**

## **::Challenge 1 (10%)**

### **Due: 11 Nov 2023, 11:59 PM**

Email to: [vu.tran@vnuk.edu.vn](mailto:vu.tran@vnuk.edu.vn)

According to the VNUK *Academic integrity policy*, plagiarism is:

*"Claiming and using the thoughts or writings or creative works of others without appropriate acknowledgement or attribution.  It includes:  
(a)  copying part or all of another student's assignment;  
(b)  allowing another person to write some or all of an assignment;  
(c)  copying paragraphs, sentences or parts of sentences directly from texts or the internet without enclosing them in quotation marks or otherwise showing them to be copied - even if the source is acknowledged, this is still plagiarism;  
(d)  using concepts or developed ideas, even if paraphrased or summarised, from another person, from texts or the internet without acknowledging the source;  
(e)  copying graphics, architectural plans, multimedia works or other forms of intellectual property without appropriate acknowledgment."*

The consequences of plagiarism (depending on the seriousness of the case) range from reducing your mark or failing the assignment up to a formal reference to a summary inquiry:

|  |
| --- |
| *By signing below I certify that the attached assignment is my own work.*  Student ID: 21040006    Student Name: Le Thi Quynh Nhu        Signature: Nhu |
|  |

**Grade:**

|  |  |  |
| --- | --- | --- |
| **No.** | **Question** | **Grade** |
| 1 | Question 1 |  |
| 2 | Question 2 |  |
| 3 | Question 3 |  |
| 4 | Question 4 |  |
| 5 | Question 5 |  |
| Total gold coins | |  |

This problem set will introduce you to using control flow in Python and formulating a computational solution to a problem.

## Data

<https://github.com/googlecreativelab/quickdraw-dataset>

The Quick Draw Dataset is a collection of 50 million drawings across [345 categories](https://github.com/googlecreativelab/quickdraw-dataset/blob/master/categories.txt), contributed by players of the game [Quick, Draw!](https://quickdraw.withgoogle.com). The drawings were captured as timestamped vectors, tagged with metadata including what the player was asked to draw and in which country the player was located. You can browse the recognized drawings on [quickdraw.withgoogle.com/data](https://quickdraw.withgoogle.com/data).



## Requirements:

|  |  |  |
| --- | --- | --- |
| **No.** | **Criteria** | **Weight (%)** |
| 1 | Train the model | 20% |
| 2 | Deploy the model | 30% |
| 3 | Explain the math/model | 15% |
| 4 | Complete app | 15% |
| 5 | Git usage | 10% |

## 

## Questions

1. **Train the model**

Choose only 2 or 4 items to train your model

Getting Data:

* creating a new CNN class for implementing a Convolutional Neural Network model
* loading three datasets (for example: car, fish and snowman)
* splitting datasets into training and test data shuffling data

Building the Model:

* creating a sequential CNN model
* adding layers to the model
* compiling the model

Training the Model

* fetching batches of data
* training, testing and evaluating the model
* plotting graphs of the model loss and accuracy during training

Reference:

<https://github.com/zaidalyafeai/Notebooks/blob/master/Sketcher.ipynb>

Explain about CNN layers:

<https://youtu.be/NL6eCtMjikQ>

1. **Deploy the model**

Use Web technology (for example Python Flask/NodeJS/PHP/Laverel) to code function same as <https://quickdraw.withgoogle.com/>

Predicting Samples

* fetching batches of samples
* predicting fetched samples

Drawing Doodles

* creating a new Painter class to allow users to draw their own doodles with the mouse
* defining painting objects: drawing area, bitmaps, pencil
* adding a function for drawing a smooth line between two points using quadratic curves

Recognizing Doodles

* resizing doodle drawing to the required size of 28x28
* normalizing array of pixels before passing it as the input of the CNN model predicting doodle

**Debriefing Report :: Part 1**

**Part 1. Report on the challenge.**

**Question 1: Train the model**

1. Items to train: Donut, door, umbrella

2. Getting Data:

* 1. Creating a new CNN class for implementing a Convolutional Neural Network model.

*model = models.Sequential()*

*model.compile(*

*loss = ‘categorical\_crossentropy”,*

*optimizer = ‘adam’,*

*metrics = [‘accuracy’]*

*)*

Explain: Organizes the model according to a standard CNN architecture, facilitating the management of layers, parameters, and training procedures.

* 1. Loading three datasets (for example: car, fish and snowman)

*def download():*

*base = 'https://storage.googleapis.com/quickdraw\_dataset/full/numpy\_bitmap/'*

*for c in classes:*

*cls\_url = c.replace('\_', '%20')*

*path = base+cls\_url+'.npy'*

*urllib.request.urlretrieve(path, 'data/'+c+'.npy')*

*download()*

Explain: The data is fetched from Google storage and stored as Numpy files, providing a convenient format for processing and use in the model.

* 1. Splitting datasets into training and test data, shuffling data

*def load\_data(root, vfold\_ratio=0.2, max\_items\_per\_class= 4000 ):*

*[…]*

*# Randomize the dataset*

*permutation = np.random.permutation(y.shape[0])*

*x = x[permutation, :]*

*y = y[permutation]*

*# Separate into training and testing based on vfold ratio*

*vfold\_size = int(x.shape[0]/100\*(vfold\_ratio\*100))*

*x\_test = x[0:vfold\_size, :]*

*y\_test = y[0:vfold\_size]*

*x\_train = x[vfold\_size:x.shape[0], :]*

*y\_train = y[vfold\_size:y.shape[0]]*

*return x\_train, y\_train, x\_test, y\_test, class\_names*

Explain: The function prepares the data for training and testing the model.

1. Building the Model:
   1. Creating a sequential CNN model

*model = models.Sequential()*

Explain: Create a CNN model using the previously defined CNN model class.

* 1. Adding layers to the model

*model.add(layers.Conv2D(32, (3, 3), padding='same', input\_shape=x\_train.shape[1:], activation='relu'))*

*model.add(layers.MaxPooling2D(pool\_size=(2, 2)))*

*model.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))*

*model.add(layers.MaxPooling2D(pool\_size=(2, 2)))*

*model.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu'))*

*model.add(layers.MaxPooling2D(pool\_size=(2, 2)))*

*model.add(layers.Flatten())*

*# Regularization and increased complexity*

*model.add(layers.Dense(256, activation='relu'))*

*model.add(layers.Dropout(0.5))*

*model.add(layers.Dense(3, activation='softmax'))*

Explain:

* Layer 1: Conv2D layer – Convolutional Layer: Convolutional layers are fundamental in CNNs for feature extraction. They apply convolutional filters to input images, enabling the model to identify patterns and spatial relationships. The parameters include:

32 filters: These filters detect different features in the input.

(3, 3) kernel size: The convolutional filter size, determining the local region analyzed.

'same' padding: Padding ensures that the spatial dimensions of the input and output remain the same.

'relu' activation: Rectified Linear Unit (ReLU) activation introduces non-linearity, allowing the network to learn complex mappings.

* Layer 2: MaxPooling2D – Max Pooling Layer: Max pooling layers downsample the spatial dimensions of the input, reducing computational complexity and focusing on the most important features. In this case, a pool size of (2, 2) is used, meaning that the maximum value in each 2x2 region is retained.
* Layer 3-6: Repeat of Convolutioncal and Max Pooling Layers: Additional convolutional and max pooling layers are added to learn increasingly complex features. The number of filters increases, allowing the model to capture more abstract representations.
* Layer 7: Flatten Layer: The Flatten layer transforms the multi-dimensional output from the previous layers into a one-dimensional vector. This is necessary when transitioning from convolutional layers to fully connected layers.
* Layer 8-9: Dense layer with regularization and Increased Complexity: Dense layers are fully connected layers where each neuron is connected to every neuron in the previous layer. Adding more neurons increases model complexity. The dropout layer introduces regularization by randomly setting a fraction of input units to zero during training, preventing overfitting.
* Layer 10: The final dense layer produces the output of the model. In this case, with three neurons and softmax activation, it outputs probabilities for each of the three classes ('donut', 'door', 'umbrella').
  1. Compiling the model

*# Compile the model*

*model.compile(*

*loss='categorical\_crossentropy',*

*optimizer='adam',*

*metrics=['accuracy']*

*)*

Explain:

* Loss Function (loss='categorical\_crossentropy'): The loss function measures the difference between the model's predictions and the actual target values. For multi-class classification problems, especially when the classes are exclusive (each instance belongs to exactly one class), categorical crossentropy is commonly used. It calculates the cross-entropy loss between the predicted probability distribution and the true probability distribution of the target.
* Optimizer (optimizer='adam'): The optimizer is responsible for adjusting the weights of the network during training to minimize the loss function. Adam (short for Adaptive Moment Estimation) is a popular optimization algorithm. It combines ideas from momentum and RMSprop to provide adaptive learning rates for each parameter, which helps converge faster and adapt to different types of data.
* Metrics (metrics=['accuracy']): Metrics are used to monitor the training and evaluation performance of the model. In this case, the accuracy metric is employed, which measures the proportion of correctly classified instances. For classification problems, accuracy is a common metric to evaluate how well the model is performing.

1. Training the Model
   1. Fetching batches of data

*# Train the model with early stopping*

*early\_stopping = tf.keras.callbacks.EarlyStopping(*

*monitor='val\_loss',*

*patience=5,*

*restore\_best\_weights=True*

*)*

Explain: Preventing Overfitting, Efficient Training, Determining the Right Number of Epochs during training

* monitor='val\_loss': specifies the metric to monitor for early stopping. In this case, the validation loss (val\_loss) is being monitored. The training process continually checks whether the validation loss improves. If the validation loss doesn't improve for a certain number of epochs (determined by patience), early stopping is triggered.
* patience=5: determines how many epochs the training process will wait for an improvement in the monitored metric before stopping. In this example, if the validation loss does not decrease for five consecutive epochs, the training will stop. This is a form of regularization that prevents overfitting; training is halted when the model starts to perform worse on unseen data (validation set).
* restore\_best\_weights=True: the model's weights are restored to the best weights observed during training. This is helpful because sometimes the model might continue training for a few more epochs before early stopping is triggered. Restoring the best weights ensures that the model has the best possible performance on the validation set when training concludes.
  1. Training, testing and evaluating the model

*history = model.fit(*

*datagen.flow(x\_train, y\_train, batch\_size=64),*

*validation\_data=(x\_test, y\_test),*

*steps\_per\_epoch=len(x\_train) // 256,*

*epochs=20,*

*callbacks=[early\_stopping]*

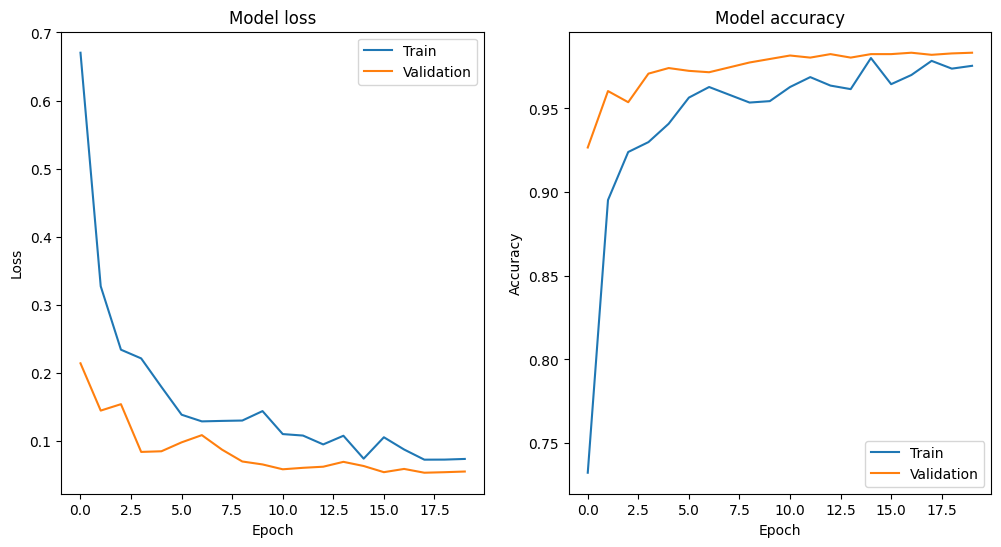
*)*

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

*print('Test accuarcy: {:0.2f}%'.format(score[1] \* 100))*

Explain:

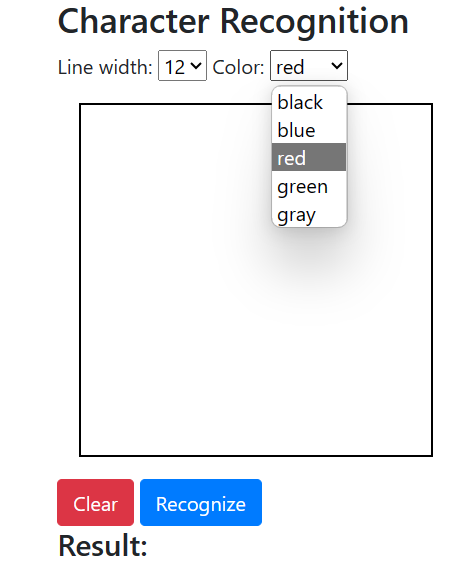
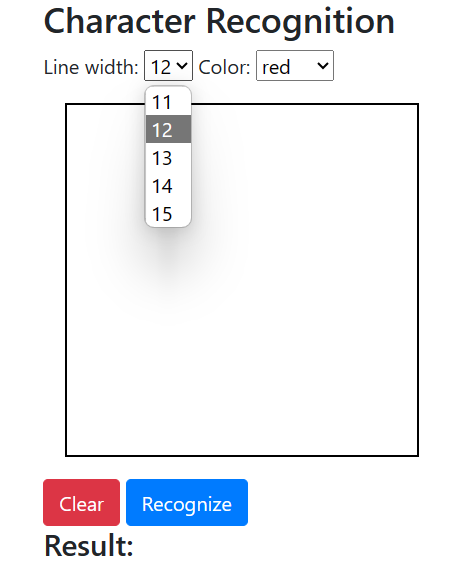
* datagen.flow(x\_train, y\_train, batch\_size=64): Generating batches of augmented data during training. datagen is an instance of ImageDataGenerator, and flow generates batches of data with real-time data augmentation. It takes x\_train and y\_train as input and processes the data in batches of size 64.
* validation\_data=(x\_test, y\_test): Specifies the data on which to evaluate the loss and any model metrics at the end of each epoch. In this case, it's using the validation set (x\_test and y\_test). The model's performance on the validation set is crucial for monitoring overfitting and generalization.
* steps\_per\_epoch=len(x\_train) // 256: Determines the number of batches to process before declaring one epoch finished and starting the next epoch. In this example, meaning that each epoch will consist of len(x\_train) // 256 batches. This useful when have a large dataset, and it allows me to process the data efficiently in chunks.
* epochs=20: Specifies the number of times the entire dataset is passed forward and backward through the neural network. In this case, the model will be trained for 20 epochs.
* callbacks=[early\_stopping]: Allows me to specify a list of callbacks to apply during training. Here, the early\_stopping callback is used to stop training early if there is no improvement in the validation loss.
  1. Plotting graphs of the model loss and accuracy during training



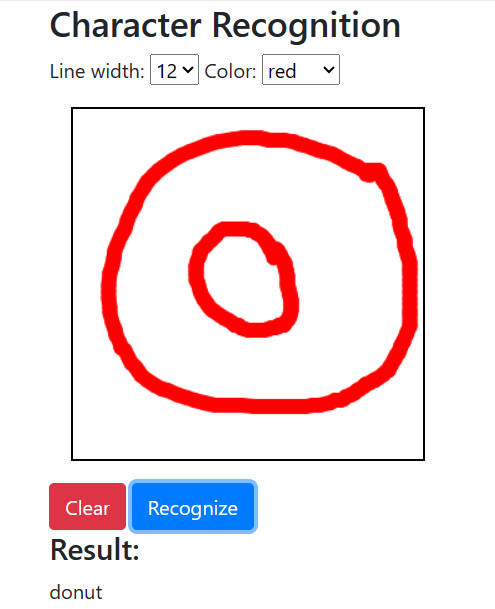
Explain: Accuracy of model increasing when number of epochs increasing and loss of model decreasing in the same period. These plots show the success in building the model CNN.

Question 2: **Deploy the model**

* + - 1. Use Web technology (for example Python Flask/NodeJS/PHP/Laverel) to code function same as <https://quickdraw.withgoogle.com/>
      2. Predicting Samples
  1. Fetching batches of samples
  2. Predicting fetched samples

1. Drawing Doodles
   1. Creating a new Painter class to allow users to draw their own doodles with the mouse
   2. Defining painting objects: drawing area, bitmaps, pencil
   3. Adding a function for drawing a smooth line between two points using quadratic curves

1. Recognizing Doodles
   1. Resizing doodle drawing to the required size of 28x28
   2. Normalizing array of pixels before passing it as the input of the CNN model predicting doodle



*References:*

*- Code and notebooks:*

*[https://github.com/zaidalyafeai/Notebooks/blob/master/Sketcher.ipynb](https://github.com/zaidalyafeai/Notebooks/blob/master/Sketcher.ipynb).*

*- Code:*

*[https://www.youtube.com/watch?v=n0k26uJR09U]*