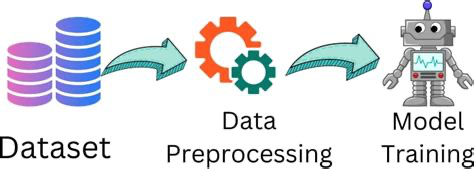
**3.1. Tiền xử lý dữ liệu đầu vào:**

Tiền xử lý dữ liệu ảnh là một bước quan trọng trong các bài toán học sâu, đặc biệt là trong các mô hình phân loại ảnh. Mục đích của tiền xử lý là chuẩn bị và làm sạch dữ liệu để giúp mô hình học tốt hơn, giảm thiểu tình trạng quá khớp (*overfitting*), đồng thời tối ưu hóa hiệu suất và tốc độ huấn luyện.



*Hình 3.1. Tiền xử lý dữ liệu đầu vào - Data Preprocessing*

Trong dự án này, dữ liệu được lấy từ thư mục dataset/rps-cv-images, gồm ba nhãn chính: Rock, Paper và Scissors. Ngoài ra, một lớp Unknown cũng được thêm vào để mô hình có thể nhận diện hình ảnh không thuộc ba nhóm trên.

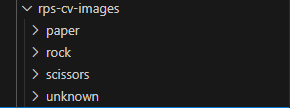
**Các bước tiền xử lý:**

* Đọc toàn bộ hình ảnh trong dataset.
* Resize hình ảnh về kích thước chuẩn (224, 224) để phù hợp với VGG19.
* Chuyển đổi giá trị pixel về khoảng [0,1] bằng cách chia cho 255.
* Gán nhãn số cho từng lớp.
* Chia dữ liệu thành tập huấn luyện (80%), tập validation (10%), và tập kiểm tra (10%).
* Lưu dữ liệu đã xử lý vào file processed\_data.npz để dùng trong quá trình huấn luyện và đánh giá mô hình.

**3.2. Dữ liệu và thư viện sử dụng:**

**3.2.1. Dữ liệu sử dụng:**

* Bộ dữ liệu được lấy từ trang web Kaggle  - “**Rock-Paper-Scissors Images**”. Bộ dữ liệu Rock-Paper-Scissors gồm ba lớp chính (Rock, Paper, Scissors).
* Thư mục Unknown chứa các hình ảnh không thuộc ba lớp trên để giúp mô hình nhận diện ảnh ngoài phạm vi.



Link dataset: <https://www.kaggle.com/datasets/drgfreeman/rockpaperscissors>

https://www.kaggle.com/datasets/pankajkumar2002/random-image-sample-dataset?resource=download

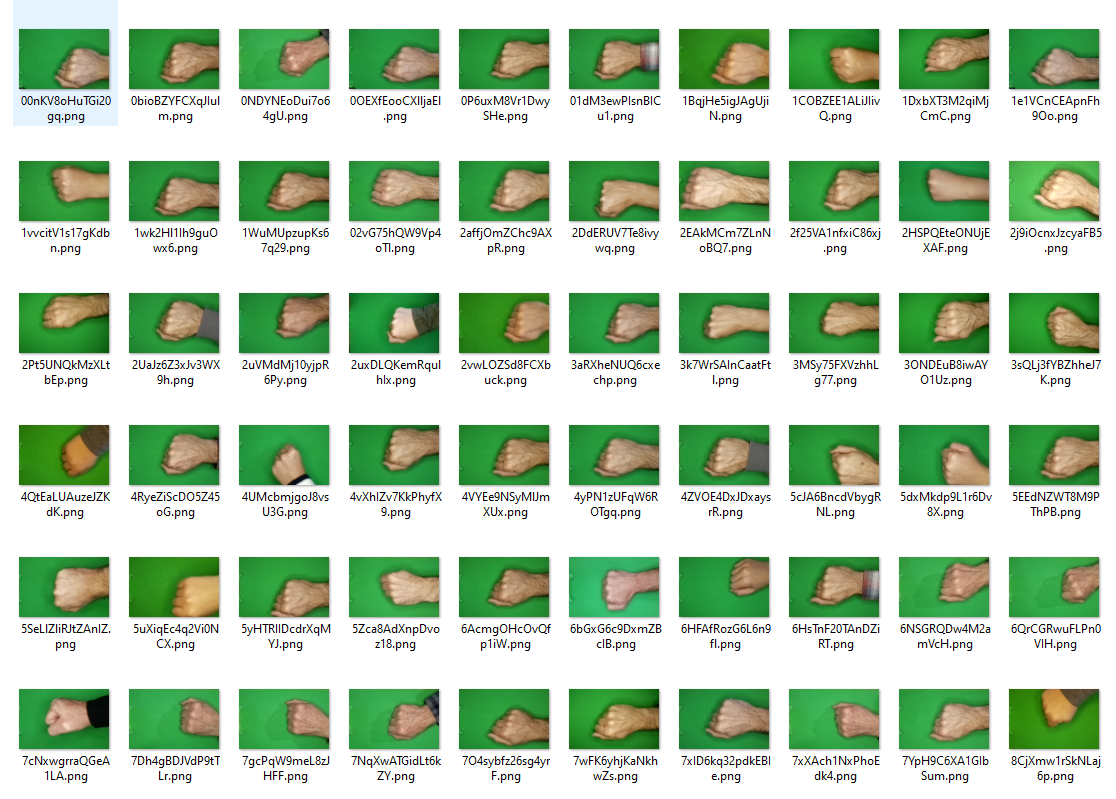
***3.2.1.1. Tập Huấn Luyện (Training Set):***

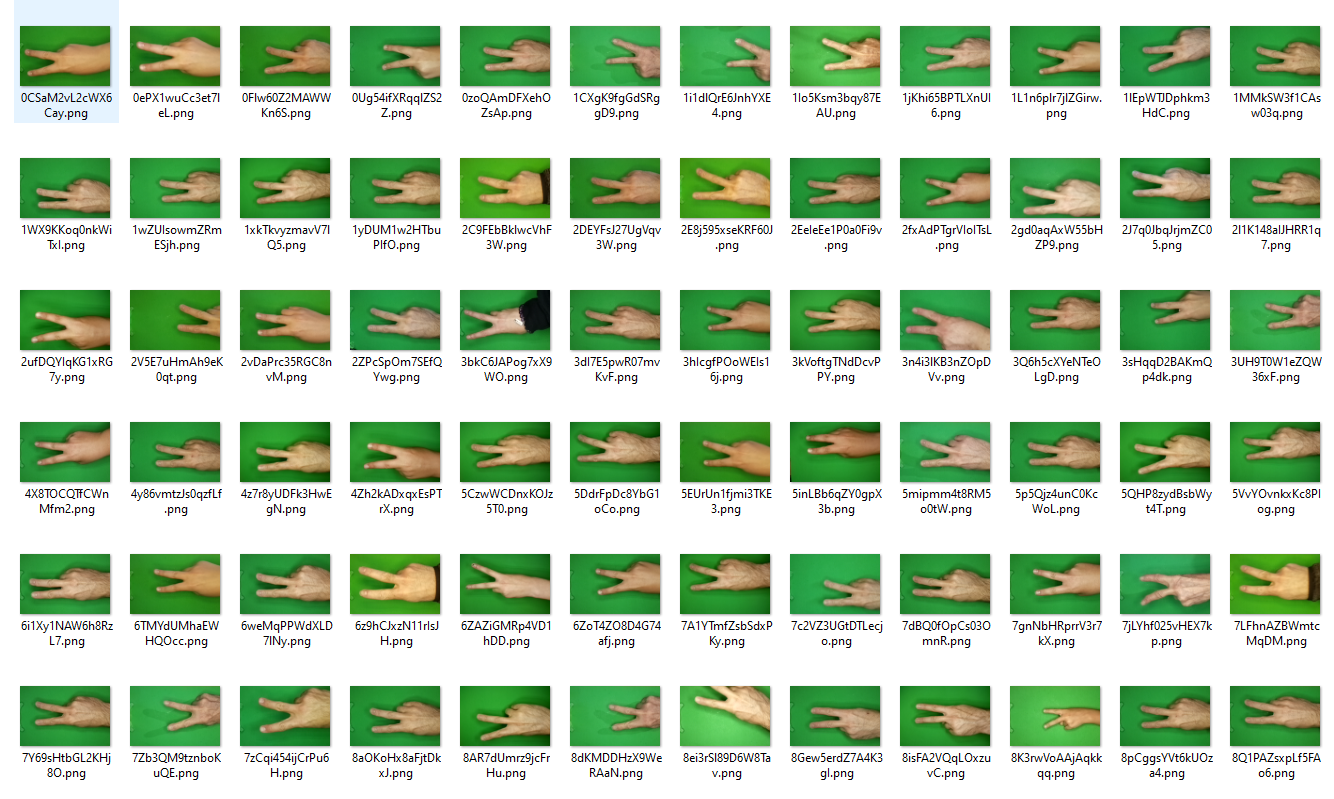
Tập huấn luyện là tập dữ liệu chính được sử dụng để huấn luyện mô hình. Mô hình sẽ học từ tập dữ liệu này để tối ưu hóa các tham số và tìm ra các đặc trưng cần thiết để phân loại hoặc dự đoán.

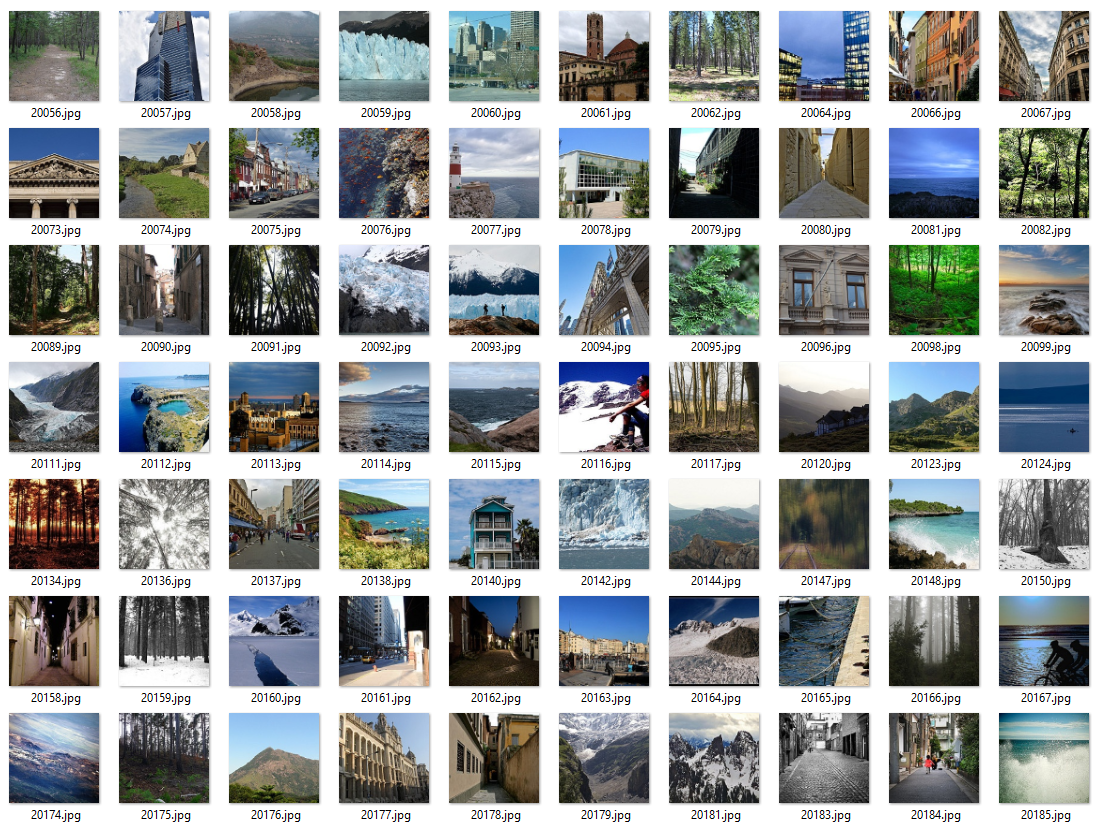
* Folder Papers: 710 ảnh



* Folder Rocks: 726 ảnh



* Folder **Scissors**: 710 ảnh 
* Folder **Unknow**: 3000 ảnh



**3.2.2. Công cụ và thư viện:**

Trong quá trình xây dựng và huấn luyện mô hình EfficientNet-B0 cho bài toán phân loại ảnh giữa chó và chó sói, ngôn ngữ lập trình Python chạy trên VS Code được sử dụng là công cụ và môi trường chính để triển khai các bước lập trình và thực hiện các phép toán phức tạp liên quan đến mô hình học sâu.



*Hình 3.4. Ngôn ngữ Python và môi trường phát triển VS Code*

Python không chỉ là một ngôn ngữ mạnh mẽ mà còn có sự hỗ trợ từ rất nhiều thư viện mạnh mẽ giúp cho việc phát triển các mô hình học máy trở nên dễ dàng và hiệu quả.

Các thư viện và công cụ hỗ trợ chính được sử dụng trong quá trình này bao gồm:

* **numpy: Xử lý dữ liệu dạng mảng.**
* **opencv-python (cv2): Đọc và xử lý ảnh.**
* **matplotlib: Vẽ biểu đồ trực quan hóa kết quả.**
* **seaborn: Hỗ trợ vẽ ma trận nhầm lẫn.**
* **tensorflow.keras: Xây dựng và huấn luyện mô hình VGG19.**
* **sklearn: Đánh giá mô hình bằng các chỉ số như classification\_report và confusion\_matrix.**
* **tkinter: Hỗ trợ chọn ảnh từ máy tính để dự đoán.**

Nhờ sự kết hợp giữa các thư viện này, quy trình huấn luyện mô hình Vgg19 được tối ưu hóa, giúp đạt được kết quả phân loại chính xác và hiệu quả trong việc phân biệt kéo búa và bao.

**3.3. Cài đặt và huấn luyện mô hình VGG19:**

**3.3.1. Cài đặt và cấu hình:**

Trong phần này, chúng ta sẽ đi qua các bước cơ bản để cài đặt và cấu hình mô hình **VGG19** trong môi trường Python.

Ta cần chạy lệnh cài đặt tất cả thư viện trong cửa sổ Terminal:

**pip install numpy opencv-python matplotlib seaborn tensorflow scikit-learn**

* **Thư viện TensorFlow & Keras** (Xây dựng mô hình deep learning)
* **OpenCV** (Tiền xử lý ảnh)
* **Matplotlib & Seaborn** (Trực quan hóa dữ liệu)
* **Scikit-learn** (Hỗ trợ đánh giá mô hình)
* **Tkinter** (Giao diện chọn ảnh để dự đoán)

Clone chương trình từ GitHub:

Sau đó ta sẽ tải Dataset về và import vào thư mục làm việc, có thể tải trực tiếp hoặc dung câu lệnh để tải   
Hướng dẫn:

**1. Cài đặt thư viện Kaggle**

Chạy lệnh sau trong terminal hoặc command prompt:

**pip install kaggle**

**2. Lấy API Key từ Kaggle**

* Truy cập [Kaggle](https://www.kaggle.com/)
* Đăng nhập vào tài khoản của bạn.
* Vào **"My Account"** (https://www.kaggle.com/account).
* Cuộn xuống phần **"API"** và nhấn **"Create New API Token"**.
* Một file kaggle.json sẽ được tải về.

**3. Cấu hình API Key**

Chuyển file kaggle.json vào thư mục phù hợp:

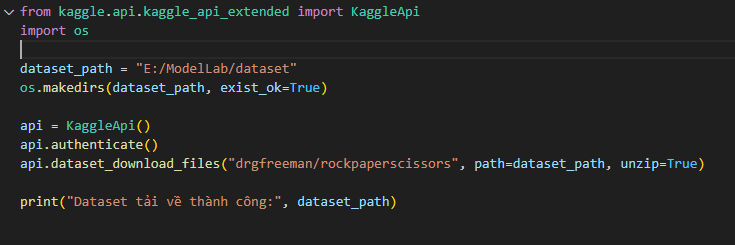
* **Windows**: Di chuyển vào C:\Users\YourUsername\.kaggle\kaggle.json
* **Linux/macOS**: Di chuyển vào ~/.kaggle/kaggle.json

Hoặc bạn có thể đặt biến môi trường:

**import os**

**os.environ["KAGGLE\_CONFIG\_DIR"] = "E:/path-to-your-kaggle-json"**

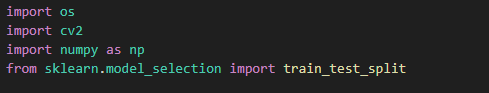
**4.Sau khi xong tạo 1 file dataset.py và chạy câu lệnh sau:**



**3.3.2. Cách thức huấn luyện mô hình:**

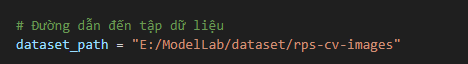
**3.3.2.1 Tiền xử lý ảnh:**

**1. Import các thư viện cần thiết**



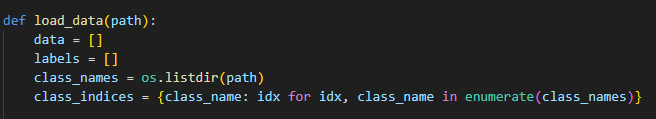
* **os: Thư viện hỗ trợ thao tác với hệ thống tệp và thư mục.**
* **cv2: Thư viện OpenCV để xử lý ảnh.**
* **numpy: Thư viện để xử lý dữ liệu dưới dạng mảng số học.**
* **train\_test\_split từ sklearn.model\_selection: Hàm dùng để chia dữ liệu thành tập huấn luyện, kiểm tra và xác thực.**

**2. Khai báo đường dẫn đến tập dữ liệu**



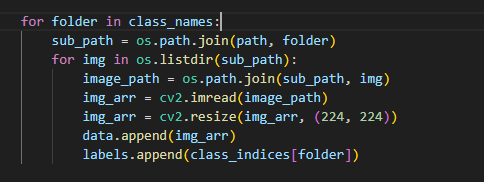
* **Đường dẫn chứa tập dữ liệu Rock-Paper-Scissors.**
* **Thư mục này gồm ba thư mục con: rock, paper, scissors, mỗi thư mục chứa ảnh tương ứng với lớp đó.**

**3. Hàm tải dữ liệu**



* **data = []: Danh sách lưu các ảnh sau khi xử lý.**
* **labels = []: Danh sách lưu nhãn (class) của ảnh.**
* **class\_names = os.listdir(path): Lấy danh sách thư mục con trong dataset\_path (tên của các lớp).**
* **class\_indices = {class\_name: idx for idx, class\_name in enumerate(class\_names)}:**
  + **Tạo từ điển ánh xạ tên lớp (rock, paper, scissors,unknown) thành số (0, 1, 2,3).**

**4. Duyệt qua từng lớp và đọc ảnh**



* **for folder in class\_names: Duyệt qua từng thư mục con (rock, paper, scissors).**
* **sub\_path = os.path.join(path, folder): Đường dẫn đến thư mục chứa ảnh của lớp hiện tại.**
* **for img in os.listdir(sub\_path): Duyệt qua từng ảnh trong thư mục con.**
* **image\_path = os.path.join(sub\_path, img): Đường dẫn đầy đủ của ảnh.**
* **img\_arr = cv2.imread(image\_path): Đọc ảnh từ đường dẫn.**
* **img\_arr = cv2.resize(img\_arr, (224, 224)): Resize ảnh về kích thước chuẩn 224×224.**
* **data.append(img\_arr): Thêm ảnh vào danh sách dữ liệu.**
* **labels.append(class\_indices[folder]): Thêm nhãn tương ứng vào danh sách nhãn.**

**5. Trả về dữ liệu dưới dạng numpy array**



* **np.array(data) / 255.0: Chuyển danh sách ảnh thành mảng NumPy và chuẩn hóa giá trị pixel về khoảng [0,1].**
* **np.array(labels): Chuyển danh sách nhãn thành mảng NumPy.**
* **class\_names: Trả về danh sách tên các lớp.**

**6. Gọi hàm load dữ liệu**



* **Gọi hàm load\_data() để đọc ảnh và nhãn từ thư mục dữ liệu.**
* **data: Mảng chứa ảnh đã chuẩn hóa.**
* **labels: Mảng chứa nhãn tương ứng.**
* **class\_names: Danh sách tên các lớp (['rock', 'paper', 'scissors']).**

**7. Chia dữ liệu thành tập huấn luyện, kiểm tra và xác thực**

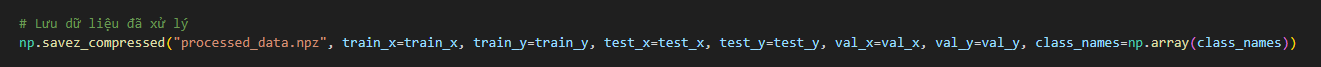


* **train\_test\_split(data, labels, test\_size=0.2, stratify=labels, random\_state=42):**
  + **Chia dữ liệu thành train (80%) và tạm thời (20%).**
  + **stratify=labels: Đảm bảo tỷ lệ giữa các lớp giống nhau trong tất cả tập dữ liệu.**
  + **random\_state=42: Đảm bảo kết quả có thể tái lập.**



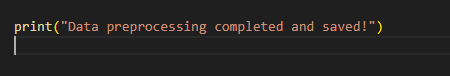
* **Chia tập tạm thời (20%) thành val (10%) và test (10%).**

**8. Lưu dữ liệu đã xử lý**



* **Lưu dữ liệu đã chia thành tập tin .npz để sử dụng sau này.**
* **train\_x, train\_y: Dữ liệu huấn luyện và nhãn.**
* **val\_x, val\_y: Dữ liệu xác thực và nhãn.**
* **test\_x, test\_y: Dữ liệu kiểm tra và nhãn.**
* **class\_names: Danh sách tên lớp.**

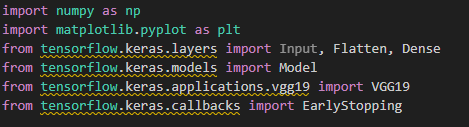
**9. Thông báo hoàn tất xử lý**



* **Xuất thông báo khi quá trình xử lý hoàn tất.**

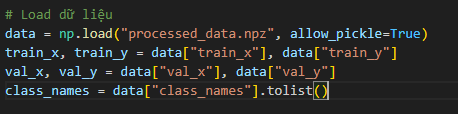
**3.3.2.2 Huẩn luyện model**

**1. Import các thư viện cần thiết**



* **matplotlib.pyplot: Thư viện để vẽ biểu đồ theo dõi quá trình huấn luyện.**
* **tensorflow.keras.layers: Các lớp cơ bản để xây dựng mô hình mạng neuron.**
  + **Input: Lớp đầu vào.**
  + **Flatten: Chuyển tensor nhiều chiều thành vector một chiều.**
  + **Dense: Lớp fully connected dùng trong mạng neuron.**
* **tensorflow.keras.models.Model: API để tạo mô hình tùy chỉnh trong Keras.**
* **tensorflow.keras.applications.vgg19.VGG19: Mô hình VGG19 tiền huấn luyện trên ImageNet.**
* **tensorflow.keras.callbacks.EarlyStopping: Dừng huấn luyện sớm nếu không có cải thiện, giúp tránh overfitting.**

**2. Tải dữ liệu đã tiền xử lý**



* **np.load("processed\_data.npz", allow\_pickle=True): Tải dữ liệu đã được lưu từ bước tiền xử lý.**
* **train\_x, train\_y: Ảnh và nhãn của tập huấn luyện.**
* **val\_x, val\_y: Ảnh và nhãn của tập validation.**
* **class\_names: Danh sách các lớp (["rock", "paper", "scissors"]).**

**3. Khởi tạo mô hình VGG19**



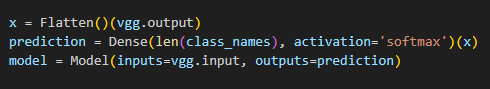
* **input\_shape=(224, 224, 3): Ảnh đầu vào có kích thước 224×224 với 3 kênh màu RGB.**
* **weights='imagenet': Sử dụng trọng số đã được huấn luyện trước trên tập dữ liệu ImageNet.**
* **include\_top=False: Bỏ lớp fully connected cuối cùng để tùy chỉnh mô hình theo bài toán mới.**

**4. Đóng băng trọng số của VGG19**



* **Duyệt qua tất cả các lớp trong VGG19 và đặt trainable = False để giữ nguyên trọng số đã được huấn luyện trước.**
* **Giúp mô hình chỉ học các lớp mới thay vì điều chỉnh toàn bộ mạng.**

**5. Thêm các lớp fully connected**



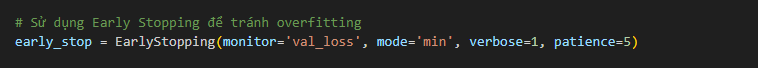
* **Flatten()(vgg.output): Chuyển tensor đầu ra từ VGG19 thành vector một chiều.**
* **Dense(len(class\_names), activation='softmax'):**
  + **Lớp đầu ra với số neuron bằng số lớp (3, tương ứng với "rock", "paper", "scissors").**
  + **activation='softmax': Biến đầu ra thành xác suất dự đoán cho từng lớp.**
* **Model(inputs=vgg.input, outputs=prediction):**
  + **Tạo mô hình với đầu vào là ảnh 224×224 RGB và đầu ra là xác suất của các lớp.**

**6. Biên dịch mô hình**



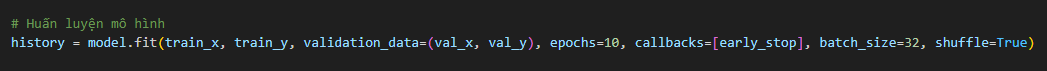
* **loss='sparse\_categorical\_crossentropy': Hàm mất mát dùng cho bài toán phân loại nhiều lớp với nhãn số nguyên.**
* **optimizer="adam": Sử dụng thuật toán Adam để tối ưu trọng số.**
* **metrics=['accuracy']: Đánh giá hiệu suất mô hình dựa trên độ chính xác.**

**7. Sử dụng Early Stopping để tránh overfitting**



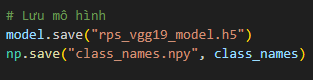
* **monitor='val\_loss': Theo dõi giá trị loss trên tập validation.**
* **mode='min': Dừng huấn luyện nếu val\_loss không giảm.**
* **verbose=1: Hiển thị thông tin khi dừng sớm.**
* **patience=5: Dừng huấn luyện nếu sau 5 epoch liên tiếp không có cải thiện.**

**8. Huấn luyện mô hình**

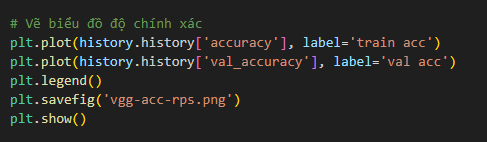
 **train\_x, train\_y: Dữ liệu huấn luyện.**

* **validation\_data=(val\_x, val\_y): Dữ liệu kiểm tra mô hình trong quá trình huấn luyện.**
* **epochs=10: Huấn luyện tối đa 10 vòng lặp.**
* **callbacks=[early\_stop]: Dừng sớm nếu cần thiết.**
* **batch\_size=32: Chia dữ liệu thành các lô nhỏ có kích thước 32.**
* **shuffle=True: Xáo trộn dữ liệu trước mỗi epoch để tránh mô hình học thứ tự cố định.**

**9. Lưu mô hình đã huấn luyện**

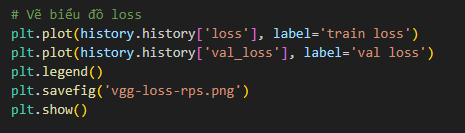


* **model.save("rps\_vgg19\_model.h5"): Lưu mô hình đã huấn luyện để sử dụng sau này.**
* **np.save("class\_names.npy", class\_names): Lưu danh sách tên lớp vào file .npy.**

**10. Vẽ biểu đồ độ chính xác**

* **history.history['accuracy'] chứa độ chính xác trên tập huấn luyện qua từng epoch.**
* **history.history['val\_accuracy'] chứa độ chính xác trên tập validation.**
* **plt.plot(...): Vẽ đồ thị độ chính xác.**
* **plt.legend(): Thêm chú thích.**
* **plt.savefig('vgg-acc-rps.png'): Lưu biểu đồ dưới dạng ảnh.**
* **plt.show(): Hiển thị biểu đồ.**

**11. Vẽ biểu đồ loss**

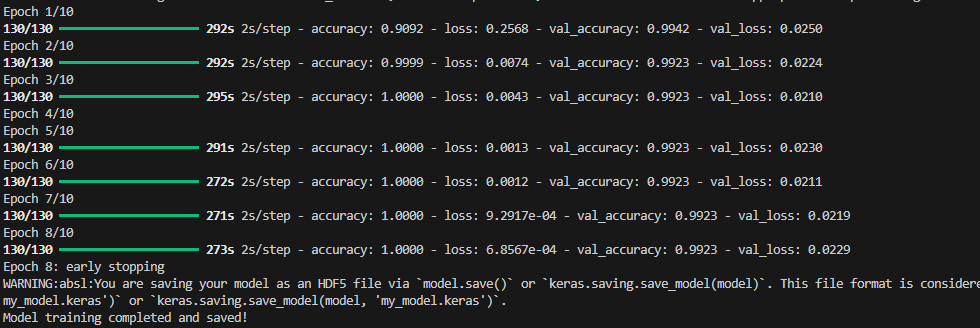


* **history.history['loss']: Hàm mất mát trên tập huấn luyện.**
* **history.history['val\_loss']: Hàm mất mát trên tập validation.**
* **Các bước còn lại tương tự biểu đồ độ chính xác.**

**12. Thông báo hoàn tất huấn luyện**

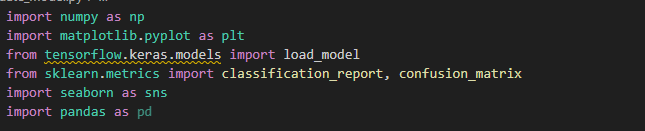


* **In ra thông báo khi quá trình huấn luyện hoàn tất và mô hình đã được lưu.**

**Quá trình chạy training:** 

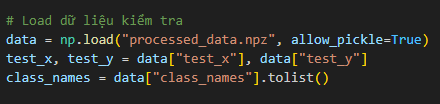
**3.3.2.3. Đánh giá model**

**1. Import các thư viện cần thiết**



* **numpy: Hỗ trợ xử lý dữ liệu dạng mảng.**
* **matplotlib.pyplot: Dùng để vẽ biểu đồ, hiển thị dữ liệu trực quan.**
* **tensorflow.keras.models.load\_model: Load mô hình đã huấn luyện trước đó để kiểm tra trên tập test.**
* **sklearn.metrics.classification\_report: Tạo báo cáo về hiệu suất mô hình (Precision, Recall, F1-score).**
* **sklearn.metrics.confusion\_matrix: Tạo ma trận nhầm lẫn để đánh giá sai số của mô hình.**
* **seaborn: Thư viện hỗ trợ vẽ biểu đồ, dùng để hiển thị ma trận nhầm lẫn một cách trực quan.**
* **pandas: Hỗ trợ xử lý dữ liệu, nhưng trong đoạn này chưa sử dụng.**

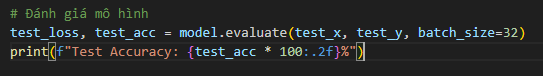
**2. Tải dữ liệu kiểm tra**



**3. Load mô hình đã huấn luyện**

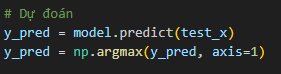


**4. Đánh giá mô hình trên tập kiểm tra**



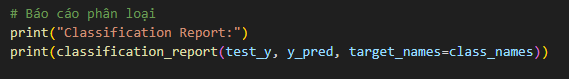
* **model.evaluate(test\_x, test\_y, batch\_size=32):**
  + **Tính toán độ chính xác (test\_acc) và hàm mất mát (test\_loss) trên tập test.**
  + **batch\_size=32: Kiểm tra theo từng nhóm 32 ảnh một lần.**
* **print(f"Test Accuracy: {test\_acc \* 100:.2f}%"):**
  + **Hiển thị độ chính xác trên tập kiểm tra dưới dạng phần trăm.**

**5. Dự đoán kết quả trên tập kiểm tra**



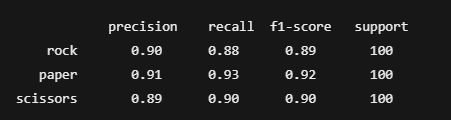
* **model.predict(test\_x):**
  + **Dự đoán xác suất của từng lớp cho mỗi ảnh trong tập test.**
  + **Kết quả là một mảng kích thước (số ảnh, số lớp), trong đó mỗi hàng chứa xác suất thuộc từng lớp.**
* **np.argmax(y\_pred, axis=1):**
  + **Chuyển xác suất thành nhãn dự đoán bằng cách chọn chỉ số có giá trị lớn nhất trên mỗi hàng.**

**6. Báo cáo phân loại**

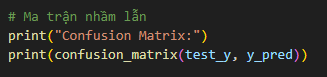


* **classification\_report(test\_y, y\_pred, target\_names=class\_names):**
  + **Tạo báo cáo chi tiết về độ chính xác (Precision), độ phủ (Recall), và điểm F1-score của từng lớp.**
  + **Precision: Tỷ lệ dự đoán đúng trên tổng số dự đoán của lớp đó.**
  + **Recall: Tỷ lệ dự đoán đúng trên tổng số mẫu thực tế của lớp đó.**
  + **F1-score: Trung bình điều hòa giữa Precision và Recall.**

**Ví dụ về đầu ra:**

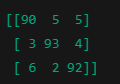


**7. Tính toán ma trận nhầm lẫn**



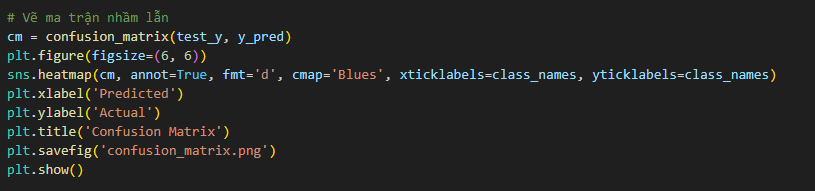
* **confusion\_matrix(test\_y, y\_pred):**
  + **Trả về ma trận nhầm lẫn cho tập test.**
  + **Hàng của ma trận là nhãn thực tế, cột là nhãn dự đoán.**
  + **Mỗi ô (i, j) thể hiện số lượng mẫu thực sự thuộc lớp i nhưng bị mô hình dự đoán thành lớp j.**

**Ví dụ về đầu ra:**



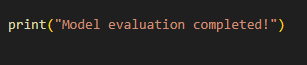
* **90 mẫu "rock" được dự đoán đúng, 5 mẫu "rock" bị nhầm thành "paper", 5 mẫu bị nhầm thành "scissors".**

**8. Vẽ biểu đồ ma trận nhầm lẫn**

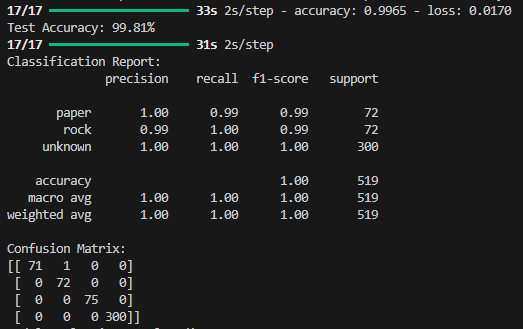


* **sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class\_names, yticklabels=class\_names):**
  + **Vẽ ma trận nhầm lẫn dưới dạng biểu đồ nhiệt (heatmap).**
  + **annot=True: Hiển thị giá trị trong từng ô.**
  + **fmt='d': Định dạng số nguyên.**
  + **cmap='Blues': Sử dụng bảng màu xanh.**
* **plt.xlabel('Predicted'): Nhãn trục X là lớp dự đoán.**
* **plt.ylabel('Actual'): Nhãn trục Y là lớp thực tế.**
* **plt.title('Confusion Matrix'): Tiêu đề biểu đồ.**
* **plt.savefig('confusion\_matrix.png'): Lưu biểu đồ vào file ảnh.**
* **plt.show(): Hiển thị biểu đồ.**

**9. Hoàn tất đánh giá**



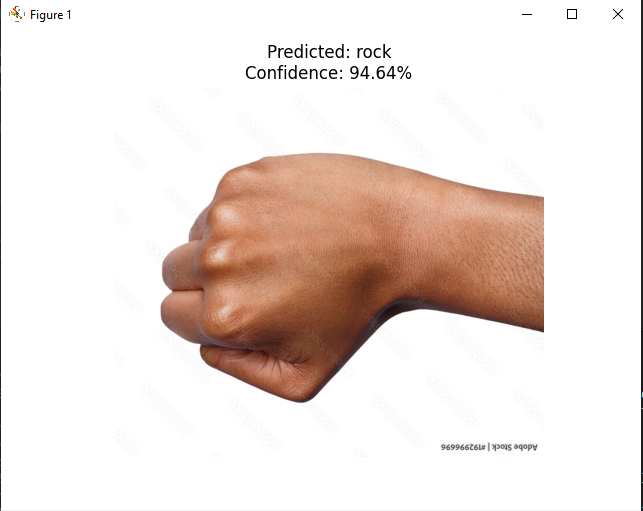
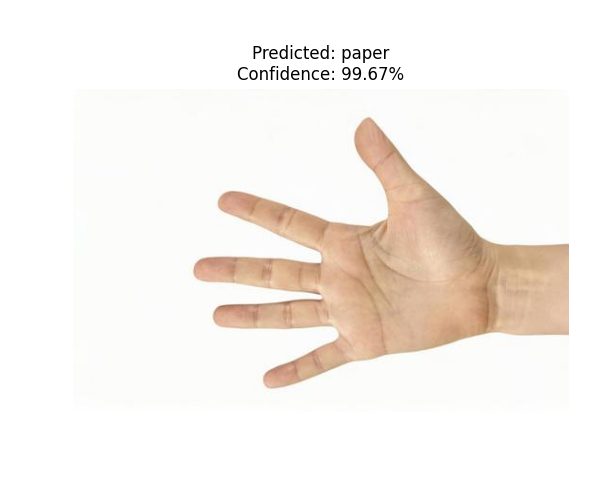
**Kết quả thực sau khi chạy:**



**3.3.3. Ứng dụng và Kết quả thực nghiệm:**

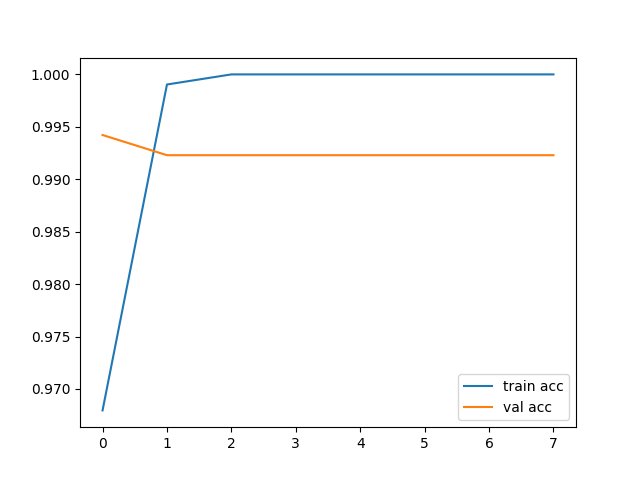
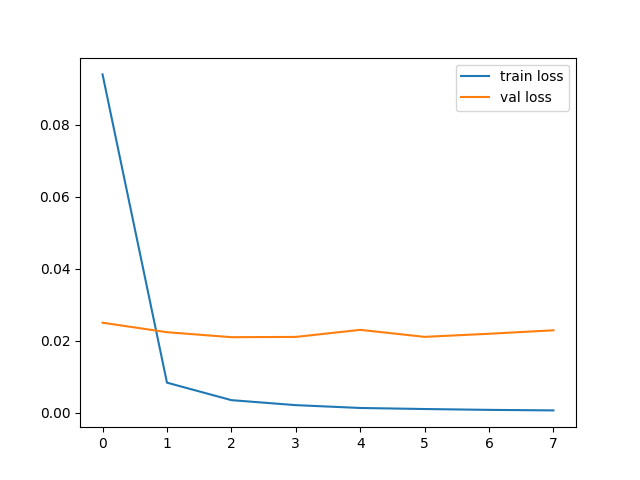
**Tạo thêm file predict.py để dự đoán kéo búa bao và kiểm tra xem model có hoạt động tốt không**

**Kết quả kiểm tra ta nhận được:**

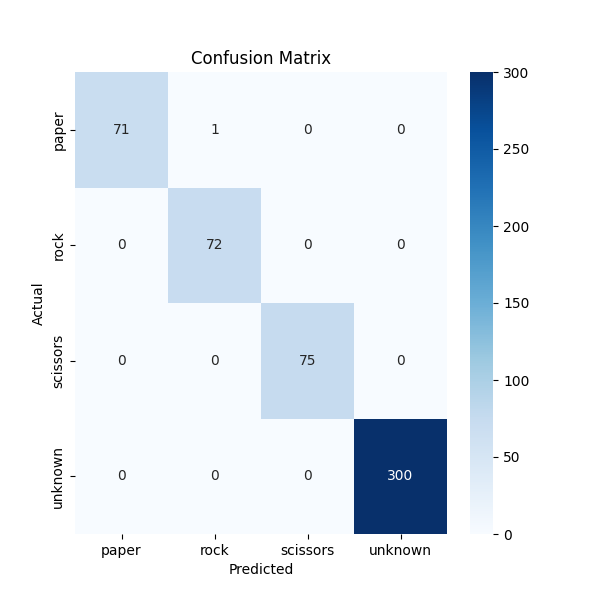
 

**3.4. Đánh giá và Phân tích kết quả:**

**3.4.1. Mô hình với Dataset tiền xử lý:**



*Biểu đồ 3.5. Loss và Accuracy*



3.6 **Ma trận nhầm lẫn dưới dạng biểu đồ nhiệt**