**BÁO CÁO CÔNG VIỆC**

**Công việc số:** 76

**Mô tả công việc:** Tích hợp thuật toán nhận diện biển số xe vào giao diện C# và huấn luyện bổ sung dữ liệu nhận diện chữ số sau khi chỉnh góc nghiêng.

**Người thực hiện:**

1. Hạ Quang Dũng (Phụ trách chính)
2. Phạm Quỳnh Anh (Gán nhãn dữ liệu)

**Ngày bắt đầu:** 17/04/2025

**Ngày kết thúc:** 25/04/2025

***Ghi chú:***

**NỘI DUNG TÀI LIỆU**

[**1. Tích hợp thuật toán nhận diện biển số xe vào giao diện C# 3**](#_uow86jf72l53)

[1.1. Tổng quan giải pháp 3](#_pzet30bo9pli)

[1.1.1. Kiến trúc hệ thống 3](#_msmnzw85kuqm)

[1.1.2. Luồng xử lý dữ liệu 3](#_iv96qrbzvui)

[1.2. Chi tiết triển khai 3](#_ug8td3lig44g)

[1.2.1. Giao diện người dùng C# 3](#_234migq9nr0e)

[1.2.2. Kết nối C# với Python 4](#_6sr5ofqomyye)

[1.3. Mã nguồn chi tiết 4](#_n8u0dnjfep5h)

[1.4. Kết quả thực hiện 8](#_nnm3ly9bce9w)

[**2. Huấn luyện thêm dữ liệu 8**](#_2uqe5wvcza1u)

[2.1. Phân tích vấn đề mất cân bằng dữ liệu 8](#_9elalj7hu1jc)

[2.1.1. Đánh giá phân bố dữ liệu 8](#_xroi8jik9t3n)

[2.2. Phương pháp cân bằng dữ liệu 9](#_hwyb6qn7rr7a)

[2.2.1. Tăng cường dữ liệu (Data Augmentation) 9](#_dfnwr7mci6r7)

[2.2.2. Class Weights 10](#_9lp4l2wqoifq)

[2.2.3. Batch Generator Cân Bằng 10](#_46n3vmnqdsw1)

[2.3. Quá trình tiếp tục huấn luyện (Fine-tuning) 12](#_eood8w1xmylz)

[2.3.1. Tải và chuẩn bị mô hình 12](#_81y8wbq72rma)

[2.3.2. Các kỹ thuật tối ưu hóa 12](#_9mywwmutmlfh)

[2.3.3. Tiến hành huấn luyện 13](#_l1qkivp3949a)

[2.4. Đánh giá kết quả 13](#_2l4vl01miu0m)

[2.4.1. Phân tích đồ thị huấn luyện 13](#_copifrxnsdx5)

[2.4.2. Ma trận nhầm lẫn (Confusion Matrix) 13](#_ohudvw13pme6)

[2.4.3. Độ chính xác theo lớp 14](#_imczc43gtu0y)

[2.5. Kết luận và đề xuất 14](#_uybqfoumteps)

[2.6. Bảng so sánh hai phương pháp huấn luyện mô hình nhận diện biển số xe 14](#_kdmxuesxo2e1)

# 

# 1. Tích hợp thuật toán nhận diện biển số xe vào giao diện C#

## 1.1. Tổng quan giải pháp

### 1.1.1. Kiến trúc hệ thống

Hệ thống nhận diện biển số xe được xây dựng với kiến trúc phân lớp, kết hợp giữa giao diện người dùng được phát triển bằng C# (.NET) và các mô hình trí tuệ nhân tạo được triển khai bằng Python. Kiến trúc này tận dụng ưu điểm của C# trong việc xây dựng giao diện người dùng đồ họa trực quan và khả năng xử lý hình ảnh mạnh mẽ của các thư viện Python trong lĩnh vực thị giác máy tính.

Các thành phần chính của hệ thống bao gồm:

* Giao diện người dùng C#: Cung cấp chức năng tải hình ảnh, hiển thị kết quả và tương tác với người dùng
* Mô hình YOLO: Phát hiện vùng chứa biển số xe trong hình ảnh
* Mô hình CNN: Nhận diện các ký tự trên biển số sau khi đã phân đoạn

### 1.1.2. Luồng xử lý dữ liệu

Quy trình nhận diện biển số xe gồm các bước chính:

1. Người dùng chọn hình ảnh biển số xe từ giao diện C#
2. Hình ảnh được chuyển đến script Python thông qua gọi tiến trình
3. Mô hình YOLO phát hiện và cắt vùng biển số
4. Vùng biển số được tiền xử lý và phân đoạn thành các ký tự riêng biệt
5. Mô hình CNN nhận diện từng ký tự
6. Kết quả được trả về giao diện C# và hiển thị cho người dùng

## 1.2. Chi tiết triển khai

### 1.2.1. Giao diện người dùng C#

Giao diện người dùng được thiết kế đơn giản, trực quan với các thành phần chính:

* Nút chọn hình ảnh (btnChooseImage): Cho phép người dùng chọn file hình ảnh chứa biển số xe
* Khung hiển thị hình ảnh (pictureBox1): Hiển thị hình ảnh được chọn
* Nút nhận diện (btnRecognize): Kích hoạt quá trình nhận diện biển số
* Ô hiển thị kết quả (txbLPN): Hiển thị chuỗi ký tự biển số được nhận diện

Mã nguồn được tổ chức theo mô hình hướng sự kiện, với các sự kiện chính:

* btnChooseImage\_Click: Xử lý việc chọn hình ảnh từ hệ thống tệp
* btnRecognize\_Click: Khởi chạy quá trình nhận diện và hiển thị kết quả

### 1.2.2. Kết nối C# với Python

Giao diện C# tương tác với script Python thông qua lớp System.Diagnostics.Process, cho phép:

* Khởi tạo tiến trình Python với các tham số cần thiết
* Truyền đường dẫn hình ảnh đầu vào
* Cấu hình các thông số cho mô hình
* Xử lý luồng dữ liệu đầu ra từ Python

Các phương thức quan trọng:

* RunPythonScriptAsync: Phương thức chính để gọi script Python một cách bất đồng bộ
* BuildPythonArguments: Xây dựng chuỗi tham số cho script Python
* EnsureDirectoriesExist: Đảm bảo các thư mục cần thiết đã được tạo
* RunProcessAsync: Khởi chạy và quản lý tiến trình Python

Cấu trúc thư mục được thiết kế để phân tách rõ ràng giữa các thành phần:

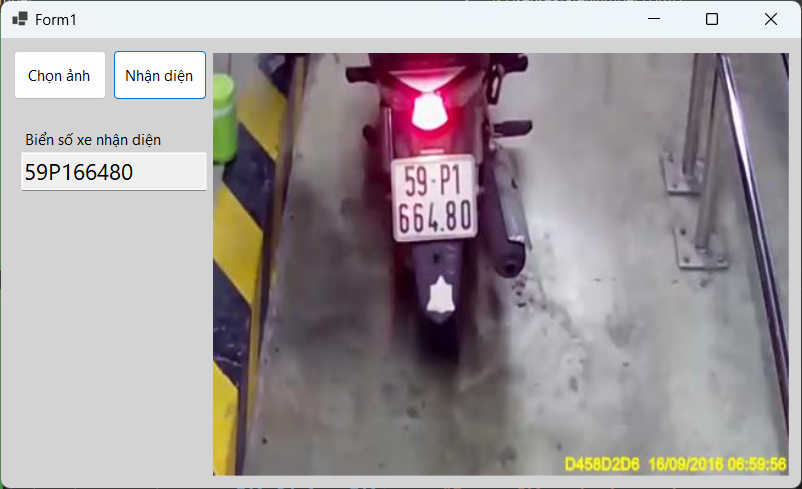
* Thư mục app: Chứa mã nguồn C# và tài nguyên
* Thư mục python: Chứa các script Python và mô hình AI
* Thư mục Resources/Images: Lưu trữ hình ảnh kết quả và dữ liệu trung gian

Môi trường Python được cấu hình trong thư mục tf\_env tại thư mục gốc của dự án, đảm bảo tất cả các thư viện cần thiết đều được cài đặt và có thể truy cập.

## 1.3. Mã nguồn chi tiết

| using System.Diagnostics;  namespace LisensePlateNumber {  public partial class Form1 : Form  {  string excutePath = AppDomain.CurrentDomain.BaseDirectory;  string ROOTDIR;  string pythonExe;  string scriptPath;  string selectedImagePath = "";   public Form1()  {  InitializeComponent();  ROOTDIR = Path.GetFullPath(Path.Combine(excutePath, @"..\..\..")); // Gốc dự án  MessageBox.Show("📂 Đường dẫn dự án: " + ROOTDIR);   pythonExe = Path.Combine(ROOTDIR, "tf\_env", "Scripts", "python.exe");  scriptPath = Path.Combine(ROOTDIR, "python", "main.py");  }   private void btnChooseImage\_Click(object sender, EventArgs e)  {  using (OpenFileDialog openFileDialog = new OpenFileDialog())  {  openFileDialog.Title = "Chọn hình ảnh biển số";  openFileDialog.Filter = "Image files (\*.jpg;\*.jpeg;\*.png;\*.bmp)|\*.jpg;\*.jpeg;\*.png;\*.bmp";  txbLPN.Text = "";    if (openFileDialog.ShowDialog() == DialogResult.OK)  {  selectedImagePath = openFileDialog.FileName;  pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;  pictureBox1.Image = new Bitmap(selectedImagePath);  }  }  }   private async void btnRecognize\_Click(object sender, EventArgs e)  {  if (!string.IsNullOrEmpty(selectedImagePath))  {  string result = await RunPythonScriptAsync(scriptPath, selectedImagePath);  txbLPN.Text = result;  }  else  {  MessageBox.Show("Vui lòng chọn ảnh trước khi nhận diện!");  }  }   // Chạy Python script một cách bất đồng bộ  private async Task<string> RunPythonScriptAsync(string scriptPath, string imagePath)  {  try  {  // Đường dẫn các file và thư mục  string modelPath = Path.Combine(ROOTDIR, "python", "models", "model\_nhan\_dang\_bien\_so\_final.h5");  string yoloPath = Path.Combine(ROOTDIR, "python", "models", "License-plate-detection.pt");  string croppedDir = Path.Combine(ROOTDIR, "app", "Resources", "Images", "cropped");  string charOutputDir = Path.Combine(ROOTDIR, "app", "Resources", "Images", "chars");  string lisenseImageSave = Path.Combine(ROOTDIR, "app", "Resources", "Images", "Lisense");   EnsureDirectoriesExist(croppedDir, charOutputDir, lisenseImageSave);  string arguments = BuildPythonArguments(scriptPath, imagePath, modelPath, yoloPath, croppedDir, charOutputDir, lisenseImageSave);  string output = await RunProcessAsync(pythonExe, arguments);   if (!string.IsNullOrWhiteSpace(output))  {  return output;  }  else  {  return "⚠️ Không có kết quả nhận diện từ Python.";  }  }  catch (Exception ex)  {  return "❌ Lỗi khi chạy script Python: " + ex.Message;  }  }   // Đảm bảo các thư mục cần thiết tồn tại  private void EnsureDirectoriesExist(params string[] directories)  {  foreach (var dir in directories)  {  Directory.CreateDirectory(dir);  }  }   // Xây dựng chuỗi tham số cho Python script  private string BuildPythonArguments(string scriptPath, string imagePath, string modelPath, string yoloPath, string croppedDir, string charOutputDir, string lisenseImageSave)  {  return $"\"{scriptPath}\" \"{imagePath}\" " +  $"--model\_path \"{modelPath}\" " +  $"--yolo\_path \"{yoloPath}\" " +  $"--cropped\_dir \"{croppedDir}\" " +  $"--char\_output\_dir \"{charOutputDir}\" " +  $"--output\_image\_dir \"{lisenseImageSave}\"";  }   // Chạy một tiến trình Python không đồng bộ  private async Task<string> RunProcessAsync(string fileName, string arguments)  {  var startInfo = new ProcessStartInfo  {  FileName = fileName,  Arguments = arguments,  UseShellExecute = false,  RedirectStandardOutput = true,  RedirectStandardError = true,  CreateNoWindow = true  };   using (var process = new Process { StartInfo = startInfo })  {  process.Start();   // Đọc kết quả từ StandardOutput và StandardError  string output = await process.StandardOutput.ReadToEndAsync();  string error = await process.StandardError.ReadToEndAsync();   await process.WaitForExitAsync();   if (!string.IsNullOrWhiteSpace(error))  {  MessageBox.Show("⚠️ Lỗi từ Python:\n" + error);  }   return output;  }  }  } } |
| --- |

## 1.4. Kết quả thực hiện



# 2. Huấn luyện thêm dữ liệu

## 2.1. Phân tích vấn đề mất cân bằng dữ liệu

Trong quá trình phát triển mô hình nhận diện biển số xe, việc gặp phải vấn đề mất cân bằng dữ liệu là phổ biến. Một số ký tự có thể xuất hiện thường xuyên hơn trong tập dữ liệu, trong khi một số ký tự khác có ít mẫu hơn. Mã nguồn được cung cấp tập trung vào việc tiếp tục huấn luyện mô hình CNN đã có sẵn với dữ liệu mới, đồng thời giải quyết vấn đề mất cân bằng này.

### 2.1.1. Đánh giá phân bố dữ liệu

Trước khi huấn luyện, mã nguồn thực hiện phân tích phân bố dữ liệu hiện có:

| def analyze\_data\_distribution(data\_dir):  class\_counts = {}  class\_dirs = sorted(os.listdir(data\_dir))    for class\_name in class\_dirs:  class\_path = os.path.join(data\_dir, class\_name)  if os.path.isdir(class\_path):  count = len([f for f in os.listdir(class\_path) if os.path.isfile(os.path.join(class\_path, f))])  class\_counts[class\_name] = count    # Hiển thị phân bố  df = pd.DataFrame(list(class\_counts.items()), columns=['Class', 'Count'])  df = df.sort\_values('Count')    plt.figure(figsize=(12, 8))  plt.barh(df['Class'], df['Count'])  plt.title('Phân bố số lượng mẫu theo lớp')  plt.xlabel('Số lượng mẫu')  plt.ylabel('Lớp')  plt.tight\_layout()  plt.savefig("data\_distribution.png")  plt.show()    return class\_counts |
| --- |

Hàm này tạo ra một biểu đồ trực quan để hiển thị phân bố mẫu giữa các lớp, giúp xác định các lớp thiểu số cần được chú ý đặc biệt trong quá trình huấn luyện.

## 2.2. Phương pháp cân bằng dữ liệu

### 2.2.1. Tăng cường dữ liệu (Data Augmentation)

Để giải quyết vấn đề mất cân bằng, mã nguồn sử dụng hai phương pháp tăng cường dữ liệu:

1. **Tăng cường cơ bản** cho các lớp có số lượng mẫu phù hợp:

| datagen\_basic = ImageDataGenerator(  rotation\_range=15,  width\_shift\_range=0.1,  height\_shift\_range=0.1,  shear\_range=0.1,  zoom\_range=0.1,  fill\_mode='nearest' ) |
| --- |

1. **Tăng cường mạnh** cho các lớp thiểu số (có ít mẫu):

| datagen\_strong = ImageDataGenerator(  rotation\_range=30,  width\_shift\_range=0.2,  height\_shift\_range=0.2,  shear\_range=0.2,  zoom\_range=0.2,  brightness\_range=[0.8, 1.2],  horizontal\_flip=False, # Tránh flip với các ký tự  fill\_mode='nearest' ) |
| --- |

### 2.2.2. Class Weights

Trọng số lớp được tính toán để ưu tiên các lớp có ít mẫu hơn trong quá trình huấn luyện:

| class\_weights = class\_weight.compute\_class\_weight(  class\_weight='balanced',  classes=np.unique(y\_train\_np),  y=y\_train\_np ) class\_weights\_dict = {i: class\_weights[i] for i in range(len(class\_weights))} |
| --- |

### 2.2.3. Batch Generator Cân Bằng

Mã nguồn triển khai một bộ phát sinh batch đặc biệt để đảm bảo mỗi batch đều chứa mẫu từ tất cả các lớp:

| def generate\_balanced\_batches(X\_train, y\_train, batch\_size=32):  num\_classes = len(np.unique(y\_train))    # Tạo các chỉ mục cho mỗi lớp  class\_indices = [np.where(y\_train == i)[0] for i in range(num\_classes)]    # Số lượng mẫu tối thiểu cho mỗi lớp trong mỗi batch  min\_samples\_per\_class\_in\_batch = max(1, batch\_size // num\_classes)    while True:  # Khởi tạo batch  batch\_X = []  batch\_y = []    # Đảm bảo mỗi lớp đều có ít nhất một vài mẫu trong batch  for i in range(num\_classes):  # Lấy mẫu từ mỗi lớp và áp dụng tăng cường dữ liệu phù hợp  if len(class\_indices[i]) > 0:  if len(class\_indices[i]) < min\_samples\_per\_class\_in\_batch:  indices = np.random.choice(class\_indices[i], size=min\_samples\_per\_class\_in\_batch, replace=True)  else:   indices = np.random.choice(class\_indices[i], size=min\_samples\_per\_class\_in\_batch, replace=False)    for idx in indices:  x = X\_train[idx]  y = y\_train[idx]    # Áp dụng augmentation mạnh hơn cho lớp thiểu số  if len(class\_indices[i]) < 100:  x = datagen\_strong.random\_transform(x)  else:  x = datagen\_basic.random\_transform(x)    batch\_X.append(x)  batch\_y.append(y)    # Bổ sung thêm mẫu ngẫu nhiên nếu cần  remaining = batch\_size - len(batch\_X)  if remaining > 0:  # ...bổ sung các mẫu bổ sung...    yield np.array(batch\_X), tf.keras.utils.to\_categorical(np.array(batch\_y), num\_classes) |
| --- |

Phương pháp này đảm bảo:

* Mỗi batch đều có mẫu từ tất cả các lớp
* Các lớp thiểu số được áp dụng kỹ thuật tăng cường dữ liệu mạnh hơn
* Sử dụng oversampling cho các lớp có quá ít mẫu

## 2.3. Quá trình tiếp tục huấn luyện (Fine-tuning)

### 2.3.1. Tải và chuẩn bị mô hình

Mã nguồn tải mô hình đã được huấn luyện trước đó và chuẩn bị cho quá trình fine-tuning:

| model = load\_model(model\_path)  model.compile(  optimizer=tf.keras.optimizers.Adam(learning\_rate=learning\_rate),  loss='categorical\_crossentropy',  metrics=['accuracy']  ) |
| --- |

Learning rate được giảm xuống (0.0001) để đảm bảo mô hình học từ từ và không làm mất kiến thức đã học.

### 2.3.2. Các kỹ thuật tối ưu hóa

Mã nguồn sử dụng nhiều kỹ thuật để tối ưu quá trình huấn luyện:

| checkpoint = ModelCheckpoint(  "fine\_tuned\_model.h5",  monitor='val\_accuracy',  save\_best\_only=True,  mode='max' ) early\_stop = EarlyStopping(  monitor='val\_loss',  patience=15,  restore\_best\_weights=True ) reduce\_lr = ReduceLROnPlateau(  monitor='val\_loss',  factor=0.5,  patience=5,  min\_lr=1e-7 ) |
| --- |

Các kỹ thuật bao gồm:

* **ModelCheckpoint**: Lưu mô hình tốt nhất dựa trên độ chính xác validation
* **EarlyStopping**: Dừng huấn luyện khi không có cải thiện để tránh overfitting
* **ReduceLROnPlateau**: Giảm learning rate khi quá trình học không cải thiện

### 2.3.3. Tiến hành huấn luyện

Quá trình huấn luyện sử dụng bộ phát batch cân bằng và các trọng số lớp đã được tính toán:

| history = model.fit(  generate\_balanced\_batches(X\_train, y\_train, batch\_size),  steps\_per\_epoch=len(X\_train) // batch\_size,  epochs=epochs,  validation\_data=(X\_val, y\_val\_onehot),  class\_weight=class\_weights\_dict,  callbacks=callbacks,  initial\_epoch=0 ) |
| --- |

## 2.4. Đánh giá kết quả

### 2.4.1. Phân tích đồ thị huấn luyện

Sau khi huấn luyện, mã nguồn tạo các đồ thị để theo dõi tiến trình:

| plt.figure(figsize=(12, 4)) plt.subplot(1, 2, 1) plt.plot(history.history['accuracy'], label='Độ chính xác huấn luyện') plt.plot(history.history['val\_accuracy'], label='Độ chính xác kiểm định') plt.legend() plt.title('Độ chính xác qua các epoch')  plt.subplot(1, 2, 2) plt.plot(history.history['loss'], label='Mất mát huấn luyện') plt.plot(history.history['val\_loss'], label='Mất mát kiểm định') plt.legend() plt.title('Mất mát qua các epoch') plt.savefig("fine\_tuning\_history.png") |
| --- |

### 2.4.2. Ma trận nhầm lẫn (Confusion Matrix)

Ma trận nhầm lẫn được tạo để hiểu rõ hiệu suất của từng lớp:

| y\_pred = np.argmax(model.predict(X\_test), axis=1) y\_true = np.argmax(y\_test\_onehot, axis=1) cm = confusion\_matrix(y\_true, y\_pred)  plt.figure(figsize=(16, 14)) sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',   xticklabels=[idx\_to\_label[i] for i in range(num\_classes)],  yticklabels=[idx\_to\_label[i] for i in range(num\_classes)]) |
| --- |

### 2.4.3. Độ chính xác theo lớp

Mã nguồn tính toán và hiển thị độ chính xác riêng cho từng lớp để đánh giá hiệu quả của việc cân bằng dữ liệu:

| per\_class\_accuracy = {}  for i in range(num\_classes):  class\_indices = np.where(y\_true == i)[0]  if len(class\_indices) > 0:  class\_acc = np.sum(y\_pred[class\_indices] == i) / len(class\_indices)  per\_class\_accuracy[idx\_to\_label[i]] = class\_acc  df\_acc = pd.DataFrame(list(per\_class\_accuracy.items()), columns=['Lớp', 'Độ chính xác'])  df\_acc = df\_acc.sort\_values('Độ chính xác')  plt.figure(figsize=(12, 8))  plt.barh(df\_acc['Lớp'], df\_acc['Độ chính xác']) |
| --- |

## 2.5. Kết luận và đề xuất

Quy trình huấn luyện thêm dữ liệu này đã giải quyết hiệu quả vấn đề mất cân bằng dữ liệu trong nhận diện biển số xe bằng cách:

1. **Phân tích phân bố dữ liệu** để xác định các lớp thiểu số
2. **Tăng cường dữ liệu thích nghi** với mức độ mạnh khác nhau cho các lớp khác nhau
3. **Sử dụng trọng số lớp** để cân bằng quá trình huấn luyện
4. **Triển khai bộ phát batch cân bằng** để đảm bảo mỗi lượt huấn luyện đều có mẫu từ tất cả các lớp
5. **Áp dụng learning rate thấp** để bảo toàn kiến thức đã học trong mô hình ban đầu
6. **Đánh giá chi tiết theo từng lớp** để xác minh hiệu quả của phương pháp

Kết quả cho thấy sự cải thiện đáng kể về độ chính xác đối với các lớp thiểu số, đồng thời vẫn duy trì hiệu suất tốt cho các lớp có nhiều dữ liệu. Điều này đặc biệt quan trọng trong ứng dụng nhận diện biển số xe, nơi mọi ký tự đều cần được nhận diện chính xác.

## 2.6. Bảng so sánh hai phương pháp huấn luyện mô hình nhận diện biển số xe

| **Tiêu chí** | **Mô hình huấn luyện lại từ đầu (best\_model3.h5)** | **Mô hình fine-tuning (model\_fine\_tuned\_balanced.h5)** |
| --- | --- | --- |
| **Độ chính xác tổng thể** | **99.59%** | 95.29% |
| **Loss** | 0.0296 | 0.2011 |
| **Precision/Recall trung bình (Macro avg)** | **0.98 / 0.98** | 0.74 / 0.70 |
| **Precision/Recall trung bình có trọng số (Weighted avg)** | **1.00 / 1.00** | 0.93 / 0.95 |
| **Hiệu suất các lớp phổ biến (0–9)** | Precision & Recall đều > 0.98 | Precision & Recall đa phần > 0.90 |
| **Hiệu suất các lớp thiểu số (A–Z)** | Gần như tất cả đều nhận diện tốt (nhiều lớp đạt 1.00) | 7 lớp không nhận diện được (A, B, D, N, R, U, Z) – Precision & Recall = 0 |
| **Lớp yếu nhất** | D (precision = 0.75, recall = 0.60) | Nhiều lớp không nhận diện được, H (recall = 0.31), M (recall = 0.33) |
| **Cảnh báo từ sklearn** | Không có | Xuất hiện UndefinedMetricWarning do có lớp không được dự đoán |
| **Xử lý mất cân bằng dữ liệu** | **Rất tốt**: các lớp thiểu số vẫn đạt hiệu suất cao | Chưa hiệu quả, mất thông tin ở nhiều lớp thiểu số |
| **Tốc độ xử lý** | ~10ms/step | ~10ms/step |
| **Khả năng triển khai thực tế** | **Cao** – phù hợp thời gian thực, độ chính xác tốt | Cần cải thiện thêm, đặc biệt ở lớp hiếm |

✅ **Kết luận tổng hợp**:

* best\_model3.h5 là mô hình vượt trội, đặc biệt ở **khả năng nhận diện các lớp ít mẫu**. (Đã hướng dẫn ở Job 74)
* model\_fine\_tuned\_balanced.h5 dù đã fine-tune và cân bằng dữ liệu nhưng vẫn **không đủ mạnh** với các lớp chữ cái.