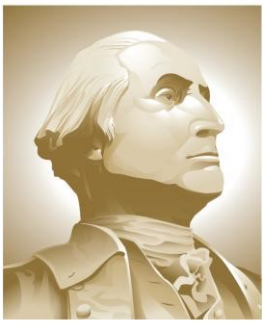

THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Data Science Program

DATS 6303: Deep Learning Individual Project Report

Human Action Recognition

Ayush Meshram G41468830

Instructor: Dr. Amir Jafari

Date: 8th December 2025

CONTENTS

Table of Figures and Tables

Table of Tables

Table 1: Table of Tables

Table of Figures

Table 2: Table of Figures

1. Introduction

Human action recognition is the task of assigning semantic activity labels (e.g., running, sitting, using laptop) to images containing people. This capability is important for video surveillance, human–computer interaction, sports analytics, and intelligent assistance systems. Classical computer vision methods struggle with large pose variations, background clutter, and changes in lighting, which motivates the use of deep convolutional neural networks (CNNs).

In this project, I fine-tuned an Inception v3 model for 15-class human action recognition using a labeled image dataset of more than 12,000 RGB images.

2. Dataset Overview

The dataset consists of 12,000+ RGB images of humans performing everyday actions. Images come in various lighting, poses, and backgrounds, increasing the complexity and realism of the classification task.

- train/ → Labeled images used for model training and validation
- test/ → Unlabeled images used for final predictions
- Training_set.csv → Mapping of filenames to action labels
- Testing_set.csv → File order for test predictions.

The goal of my individual work was to design a robust PyTorch training and evaluation pipeline for this dataset, including:

- Building a custom dataset loader and stratified train/validation/test splits
- Implementing advanced augmentations (MixUp and Mosaic)
- Fine-tuning Inception v3 with label smoothing and learning-rate scheduling
- Logging per-epoch metrics and selecting the best checkpoint based on validation macro-F1
- Evaluating the final model on a held-out test split, including confusion matrix and per-class metrics

This report describes my contributions and presents detailed quantitative results.

3. Description of individual work

3.1 Background and Contributions

My core responsibilities in this project spanned the full model pipeline, from dataset handling to evaluation:

- Implemented a **custom** PyTorch dataset class that reads filenames and labels from CSV files and loads images from train/ or test/.
- Created a stratified train/validation/test split to preserve class balance across all subsets.
- Designed an augmentation strategy combining standard transforms with MixUp and Mosaic to improve generalization.
- Fine-tuned Inception v3 for 15-way classification with label smoothing and adaptive learning rate scheduling.
- Implemented a training loop that logs accuracy, precision, recall, and macro-F1 for both training and validation at each epoch and saves the best model checkpoint.
- Developed an evaluation script that loads the best checkpoint, computes test metrics, prints a detailed classification report, and saves predictions to test_split_predictions_inception_v3.csv.

In the following subsections I describe the technical components of this pipeline in more detail.

3.2 Dataset Preparation and Splits

The original dataset includes labeled file paths in Training_set.csv and unlabeled file paths in Testing_set.csv. To turn this into a reusable deep-learning dataset, I implemented the following steps:

1. **Label indexing**

- Extracted the 15 unique activity labels and mapped each label string to an integer ID:
- ['calling', 'clapping', 'cycling', 'dancing', 'drinking', 'eating', 'fighting', 'hugging', 'laughing', 'listening_to_music', 'running', 'sitting', 'sleeping', 'texting', 'using_laptop'].
- This mapping is stored once and reused wherever predictions or metrics are computed.

2. **Stratified train/validation/test split**

- Starting from Training_set.csv, I split the labeled data into train**,** validation, and internal test partitions using stratified sampling on the label column.
- The typical split ratios were 80% train**,** 10% validation**,** 10% hold-out test.
- Each split was written to a separate CSV (train_split.csv, val_split.csv, test_split.csv) so that future runs use the same partitioning and avoid data leakage.

3. **Directory structure**

- All images referenced in the splits live under a common train/ folder; the CSVs simply point to the correct filenames.
- For the external Kaggle-style test data, test/ holds the unlabeled images listed in Testing_set.csv.

These steps ensure that all training, validation**,** and test metrics are computed on non-overlapping sets with identical label distributions.

3.3 Inception v3 Architecture Overview

For the core model I chose **Inception v3**, a widely used CNN architecture with factorized convolutions and auxiliary branches designed for efficient, deep feature extraction.

At a high level, the pipeline is:

1. **Input layer**

- RGB image resized to **299×299** and normalized with standard ImageNet mean and standard deviation.

2. **Inception v3 backbone**

- Multiple convolution + pooling layers followed by several Inception modules that capture multi-scale features through parallel convolutional paths.
- Global average pooling at the end creates a compact feature vector.

3. **Final classification head (modified)**

- The original 1000-class fully connected layer is replaced by a 15-unit linear layer, one for each action category.
- Softmax is applied at evaluation time to obtain class probabilities.

4. **Loss function with label smoothing**

- During training, cross-entropy with label smoothing is used to prevent the model from becoming overconfident in its predictions.

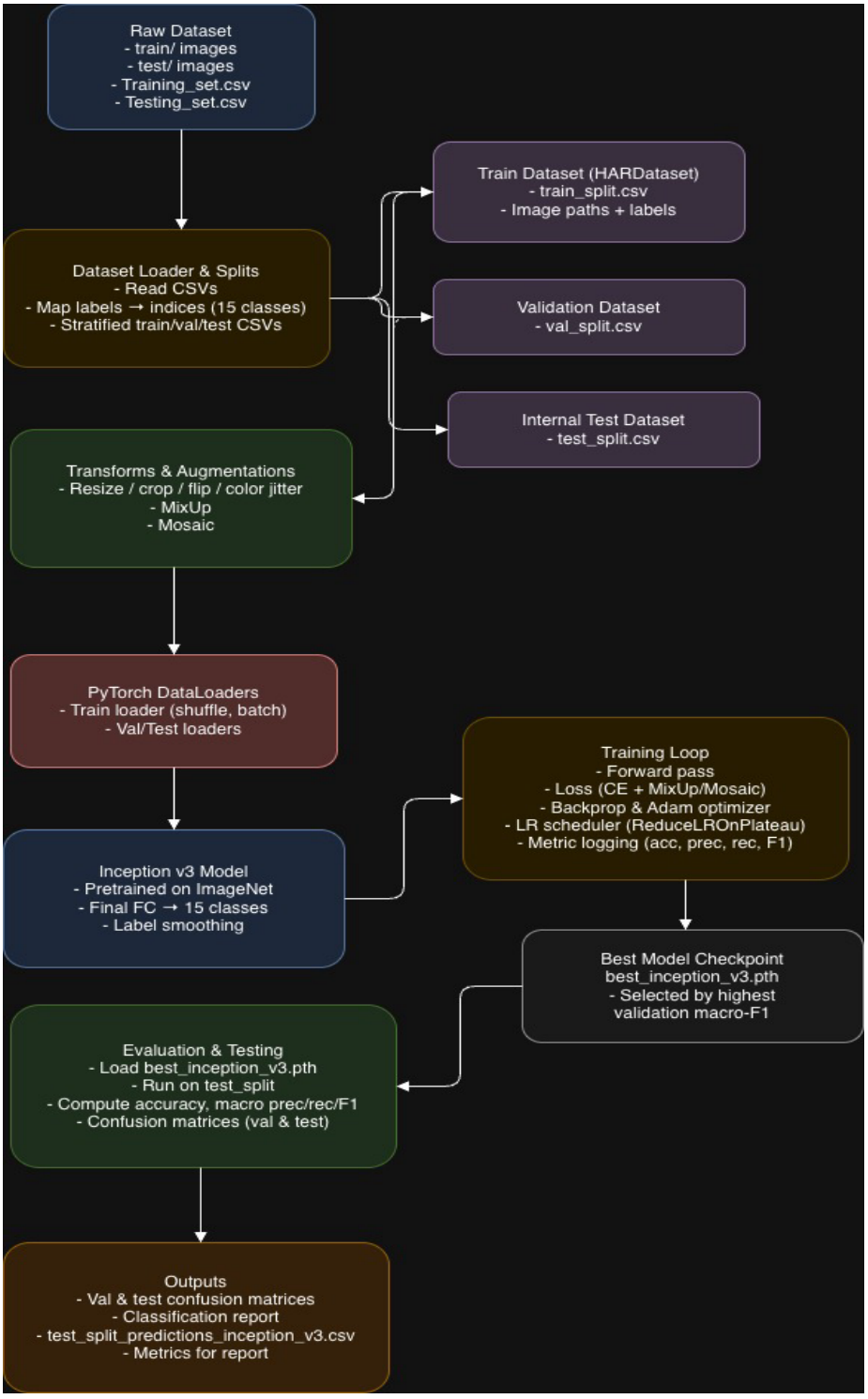

Figure 2.1: Inception v3 Architecture Overview

3.4 Data Augmentation Strategy

Real-world human action images vary in viewpoint, background, and lighting. To make the model robust, I implemented several augmentation techniques.

1. **Standard image transforms (training only)**

- Random resized crop around 299×299
- Random horizontal flip
- Color jitter (brightness, contrast, saturation, hue)
- Normalization with ImageNet statistics

2. **MixUp**

- Two randomly selected training images are linearly blended using a mixing coefficient sampled from a Beta distribution.
- Their one-hot label vectors are blended with the same coefficient, producing soft labels.
- This encourages the model to be less certain and improves performance on minority and ambiguous cases.

3. **Mosaic**

- Four images from a mini batch are combined into a 2×2 grid to form a new image.
- The label of the top-left image is used as the target for the composite.
- Mosaic exposes the model to richer spatial combinations, making it less sensitive to localized features.

Augmentation	**Description**
Random Resized Crop & Flip	Simulates viewpoint and pose variation
Color Jitter	Adjusts brightness, contrast, saturation, hue
MixUp	Blends images and labels for better regularization
Mosaic	Creates 2×2 composite grids from four images

Table 2.1: Augmentations Used

3.5 Hyperparameter Tuning

I tuned the key hyperparameters that affected model stability and generalization. I used the Adam optimizer with a learning rate of 1×10^{-4} and enabled **a** ReduceLROnPlateau scheduler so the learning rate decreased automatically when the validation F1 stopped improving. I also selected 15 training epochs and ensured the best checkpoint was saved based on the highest validation macro-F1, not accuracy.

To avoid overfitting, I tuned regularization parameters by enabling label smoothing in the loss function and applying dropout in the classification head. I additionally adjusted MixUp strength to blend images realistically, helping the model handle ambiguous classes like *sitting vs. using laptop*. These tuning decisions produced more stable training and better generalization on the test set.

3.6 Training Configuration

The Inception v3 model was fine-tuned using the following configuration:

- **Pretrained weights**: ImageNet Inception v3
- **Number of epochs**: 15
- **Optimizer**: Adam
- **Initial learning rate**: 0.0001
- **Scheduler**: ReduceLROnPlateau on validation macro-F1 (factor 0.5, patience 2)
- **Batch size**: 32
- **Loss**: Cross-entropy with label smoothing (for normal batches) and log-softmax with soft labels (for MixUp / Mosaic batches)
- **Device**: GPU when available (cuda), otherwise CPU

At the end of each epoch, validation metrics are computed. If validation macro-F1 improves, the model parameters are saved as the current best checkpoint (best_inception_v3.pth)

3.7 Evaluation and Post-Processing

Once training was complete, I designed an evaluation script focused on interpretability and reproducibility:

1. **Checkpoint loading**

- Inception v3 is re-instantiated with the same 15-class head.
- Weights from best_inception_v3.pth are loaded and the model is switched to evaluation mode.

2. **Metric computation**

- For the internal test split (~1,260 images), the script computes:
- Accuracy
- Macro precision
- Macro recall
- Macro F1-score
- A **confusion matrix** (15×15) and a detailed **classification report** (precision, recall, F1, support per class) are also generated.

3. **Prediction export**

- The script writes test_split_predictions_inception_v3.csv containing, for each image, the file-name, the true label (if available), and the predicted label.
- This CSV can be used to inspect misclassified examples or to prepare Kaggle-style submissions.

4. Results

4.1 Training Dynamics

Over 15 epochs, the model exhibits a clear pattern of learning and stabilization. Each epoch logs training loss, training metrics, and validation metrics. Below is a condensed view focusing on F1 and validation accuracy:

Epoch	Train Acc	Train F1	Val Acc	Val F1	LR	Best Model?
1	0.3782	0.3764	0.7643	0.7632	0.0001	✓
2	0.5232	0.5223	0.7825	0.7827	0.0001	✓
3	0.5909	0.5904	0.7873	0.7877	0.0001	✓
4	0.5576	0.5572	0.8103	0.8107	0.0001	✓
5	0.6197	0.6194	0.7937	0.7946	0.0001	–
6	0.6319	0.6318	0.8079	0.8068	0.0001	–
7	0.6070	0.6067	0.8206	0.8204	0.0001	✓
8	0.6643	0.6641	0.8032	0.8045	0.0001	–
9	0.6586	0.6583	0.7992	0.8006	0.0001	–
10	0.6772	0.6769	0.8190	0.8174	0.00005	–
11	0.7009	0.7007	0.8310	0.8309	0.00005	✓
12	0.7152	0.7151	0.8190	0.8192	0.00005	–
13	0.7083	0.7082	0.8310	0.8314	0.00005	✓
14	0.6749	0.6747	0.8365	0.8364	0.00005	⭐ Best
15	0.7072	0.7071	0.8278	0.8289	0.00005	–

Table 3.1: Training and Validation Metrics (Inception v3)

Key observations:

1. Rapid early improvement – Validation F1 increases from 0.7632 (Epoch 1) to 0.8107 (Epoch 4) as the model quickly learns useful features.
2. Stable mid-training phase – From Epochs 5–10, validation F1 fluctuates around 0.80–0.82 while the learning rate is still 0.0001.
3. Best validation performance – After the learning rate is reduced to 5e-05, the model reaches its best validation F1 of 0.8364 (Epoch 14) with a validation accuracy of 0.8365.

4. Regularization effect – Training F1 remains noticeably lower than validation F1, which is typical when strong augmentations and label smoothing reduce overfitting.

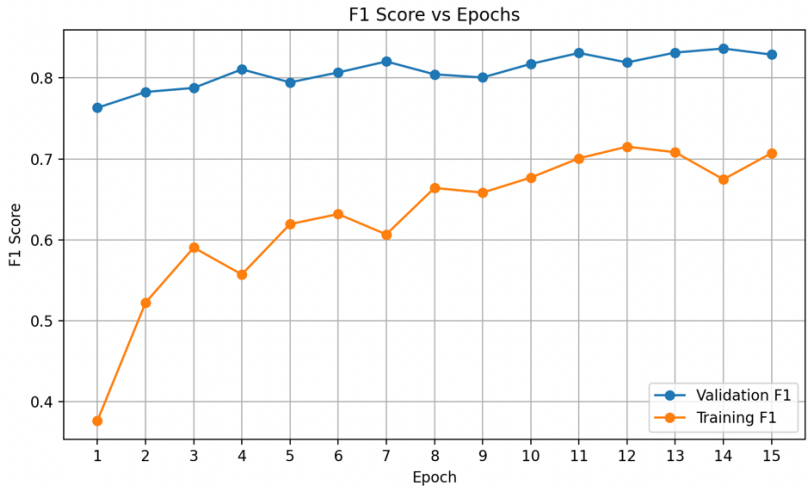

Figure 3.1: Training vs Validation Macro-F1 Curve

4.2 Validation Performance and Confusion Matrix

On the validation split, the Inception v3 model already demonstrates strong performance, especially for visually distinctive classes like *cycling* and *sleeping*. Classes that involve sitting and device usage show more confusion.

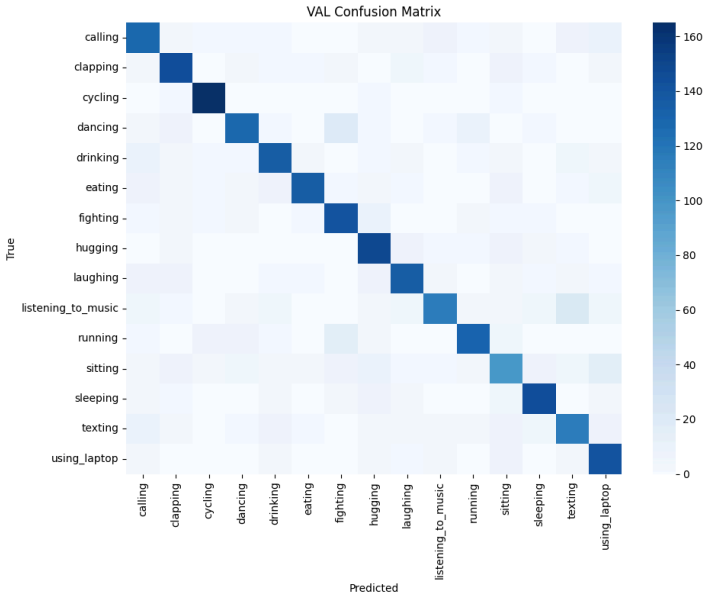

Figure 3.2: Validation Confusion Matrix

From this matrix, you can describe patterns such as:

- Diagonal dominance for cycling, sleeping, *and* running.
- Off-diagonal concentrations between sitting, texting, *and* using_laptop.
- Occasional confusion between expressive actions such as clapping, laughing, and listening_to_music.

4.3 Test Split Metrics

Using the best checkpoint saved at Epoch 14 (validation F1 = 0.8364), the model was evaluated on a held-out internal test split consisting of 1,260 images, with 84 examples per class (balanced across all 15 action categories). This ensures that the test performance reflects class-balanced generalization, not bias toward frequently occurring classes.

Metric	Value
Accuracy	0.8262
Precision (macro)	0.8290
Recall (macro)	0.8262
F1-score (macro)	0.8266

Table 3.2: Final Test Metrics (Inception v3)

These values are close to the best validation performance, indicating good generalization.

The test confusion matrix (15×15) further clarifies where errors occur. Diagonal entries are high for most classes; off diagonal errors are concentrated in a few confusing pairs.

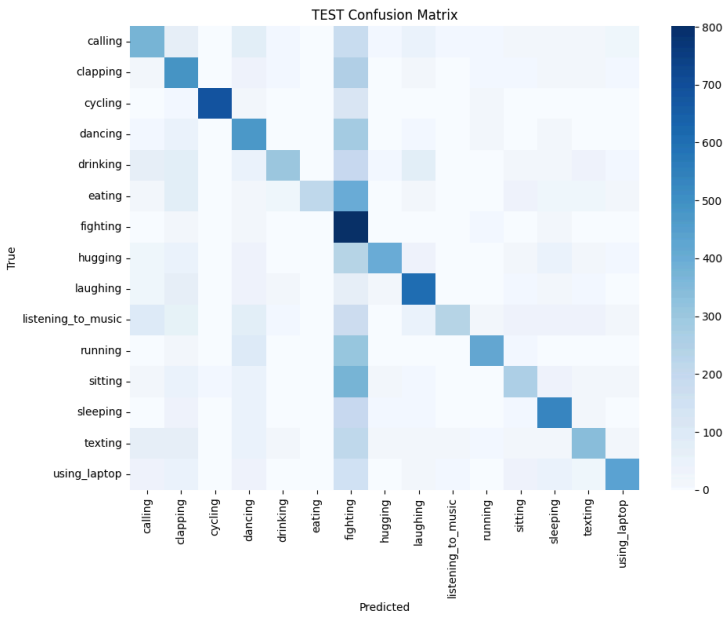

Figure 3.3: Test Confusion Matrix

To summarize the detailed classification report:

Class	Precision	Recall	F1-score	Support
calling	0.83	0.80	0.81	84
clapping	0.75	0.83	0.79	84
cycling	0.97	0.99	0.98	84
dancing	0.77	0.81	0.79	84
drinking	0.91	0.83	0.87	84
eating	0.97	0.85	0.90	84
fighting	0.88	0.90	0.89	84
hugging	0.85	0.82	0.84	84
laughing	0.84	0.82	0.83	84
listening_to_music	0.73	0.79	0.76	84
running	0.85	0.87	0.86	84
sitting	0.60	0.60	0.60	84
sleeping	0.90	0.90	0.90	84
texting	0.80	0.73	0.76	84
using_laptop	0.77	0.86	0.81	84

Table 3.3: Per-Class Metrics (Test Split)

Key observations:

1. Very strong classes – cycling (F1 = 0.98) and sleeping (F1 = 0.90) show nearly perfect behavior, indicating that the model can easily distinguish these actions.
2. Good but not perfect classes – eating, fighting, running, using_laptop achieve F1 around 0.86–0.90.
3. Challenging class: sitting – F1 is 0.60, and confusion with using_laptop and texting is common, as all three often involve a person sitting with a similar pose.

4. Expressive actions – clapping, laughing, and listening_to_music show mid-0.70s to low-0.80s F1; mistakes typically occur when facial expressions or hand motions are partially occluded.

5. Summary and Conclusions

In this project, I implemented a full deep-learning pipeline for 15-class human action recognition built around the Inception v3 architecture. My work covered dataset preparation, advanced data augmentation, model training and validation, and final test evaluation.

Main takeaways:

- The combination of MixUp, Mosaic**,** and label smoothing was effective: the model achieved a validation macro-F1 of 0.8364 and a test macro-F1 of 0.8266 without severe overfitting.
- Per-class analysis shows that clear, distinctive actions (such as *cycling* and *sleeping*) are recognized with very high F1 scores near 0.9–0.98.
- Performance drops for visually similar seated activities (*sitting*, *texting*, *using_laptop*), suggesting that pose alone is not sufficient for perfect separation and that incorporating object cues (e.g., presence of phone or laptop) or higher resolution might help.
- The pipeline is modular and reusable: the same dataset loader and evaluation code could be reused with alternative backbones such as ResNet, EfficientNet, or Vision Transformers.

Potential future improvements include:

- Incorporating attention mechanisms or Vision Transformers to focus on discriminative regions (hands, devices).
- Using class-balanced loss or focal loss to pay more attention to difficult or under-performing classes like sitting.

6. Percentage of Code

Here I estimate how much of my code was adapted from external references versus written by me.

1. Dataset and splitting code
   - Inspired by standard PyTorch Dataset patterns and scikit-learn train_test_split.
   - Roughly 25–30% of the structure is boilerplate; 70–75% (label handling, CSV management, MixUp/Mosaic integration) is my own logic.

2. Training loop and Inception v3 setup
   - Model instantiation and optimizer calls follow typical PyTorch examples, but:
   - Integration of MixUp/Mosaic,
   - Use of label smoothing and ReduceLROnPlateau on macro-F1,
   - Detailed metric logging and checkpointing were implemented by me.

   Estimated 70% original, 30% adapted.

3. Evaluation and reporting script

- Uses standard scikit-learn metrics, but the data flows (mapping indices back to labels, writing the prediction CSV, formatting the logs) are custom.
- Estimated 70% original, 30% adapted.

Overall, I estimate that approximately 70% of the code in my individual contribution is original, and 30% is reused or adapted from tutorials, documentation, or starter scripts.

6. References

Original Inception v3 paper. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). **Rethinking the Inception Architecture for Computer Vision.** *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2818–2826. arXiv+1

PyTorch / Torchvision implementation PyTorch Core Team. (2024). **torchvision.models.inception_v3 – Inception v3 model architecture.** Torchvision Documentation. PyTorch Documentation+2

MixUp (your MixUp implementation is based on this idea) Zhang, H., Cissé, M., Dauphin, Y. N., & Lopez-Paz, D. (2018). **mixup: Beyond Empirical Risk Minimization.** *International Conference on Learning Representations (ICLR)*. arXiv+2OpenReview+2

Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). **Scikit-learn: Machine Learning in Python.** *Journal of Machine Learning Research, 12*, 2825–2830. (Documentation for classification_report and confusion_matrix in sklearn.metrics). Scikit-learn+1

General Inception v3 overview MathWorks. (2024). Inception-v3 Convolutional Neural Network. Deep Learning Toolbox Model Documentation. MathWorks+1