Image Classification Using Convolutional Neural Networks (CNN)

NeuralLens Group:

Renad Almaghthawi

Lina Alshammari

Reema Alabisi

Table of contents

01
Introduction

Introduction

In this project, we used two approaches to build the CNN model: one from scratch and the other using Transfer Learning.

The scratch-based approach allowed us to design and train the model from the ground up, while Transfer Learning enabled us to leverage pre-trained models to enhance performance and reduce training time.

02
Animals-10 Dataset

Animal Dataset

The Animal Dataset is a well-labeled and diverse collection of animal images, available on Kaggle. It is ideal for projects focused on image classification. this dataset provides a variety of opportunities. Below are some key reasons why this dataset is a great choice for our project:

- **Diverse Categories:** Contains 10 different animal categories (cats, dogs, rabbits, etc.).
- **Model Challenges:** It provides challenges in improving the model and reducing overfitting, making it a good training ground for model optimization.
- **Images quality: It** is moderate but relatively good compared to CIFAR-10 dataset
- **Number of images:** It have 28,000 images is lower compared to CIFAR-10 dataset

Preprocess the data

- **Resize Images:** Since the images in the dataset have different sizes, they must be resized to a consistent shape (224x224) to ensure that the model can process them effectively.

- **Normalize Pixel Values:** To improve the performance of the neural network, the pixel values are scaled from the range [0, 255] to [0, 1].

- **Convert Labels to One-Hot Encoding:** Since the model only accepts numerical data, labels are converted into one-hot encoded vectors.

- **Split Data into Train and Test Sets:** This ensures an accurate evaluation of the model's performance by training on one subset of the data and testing on another.

- **Image Augmentations:** Use ImageDataGenerator with augmentations to enhance the training dataset by applying transformations like rotation, shifting, and flipping.

02

CNN Model Development

Build the first CNN model from scratch

1. CNN-based architecture (simple)
2. CNN Architectures :
   - VGG16
   - LeNet-5
   - AlexNet

CNN Architectures :

- **CNN-based architecture** : A simple convolutional neural network with two Conv2D layers, max-pooling, and dense layers for classification. Ideal for small datasets or educational purposes.
- **LeNet-5 :** A classic CNN architecture with 7 layers, originally designed for handwritten digit recognition. Suitable for small-scale classification tasks.
- **AlexNet** : A deep CNN architecture with 8 layers, introduced ReLU activation, and dropout for preventing overfitting. It’s designed for large-scale image classification tasks.
- **VGG16** : A deep CNN model with 16 layers, using small 3×3 convolutions and max-pooling. Known for its uniform design and effective in large-scale image classification tasks.

CNN-based architecture:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
flatten (Flatten)	(None, 394272)	0
dense (Dense)	(None, 100)	39,427,300
dense_1 (Dense)	(None, 10)	1,010

Total params: 39,429,208 (150.41 MB)
Trainable params: 39,429,206 (150.41 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 2 (12.00 B)

CNN Architecture : VGG16

Model: "sequential_5"

Layer (type)	Output Shape	Param #
conv2d_95 (Conv2D)	(None, 224, 224, 64)	1,792
batch_normalization_94 (BatchNormalization)	(None, 224, 224, 64)	256
conv2d_96 (Conv2D)	(None, 224, 224, 64)	36,928
max_pooling2d_5 (MaxPooling2D)	(None, 112, 112, 64)	0
conv2d_97 (Conv2D)	(None, 112, 112, 128)	73,856
batch_normalization_95 (BatchNormalization)	(None, 112, 112, 128)	512
conv2d_98 (Conv2D)	(None, 112, 112, 128)	147,584
max_pooling2d_6 (MaxPooling2D)	(None, 56, 56, 128)	0
conv2d_99 (Conv2D)	(None, 56, 56, 256)	295,168
batch_normalization_96 (BatchNormalization)	(None, 56, 56, 256)	1,024
conv2d_100 (Conv2D)	(None, 56, 56, 256)	590,080
max_pooling2d_7 (MaxPooling2D)	(None, 28, 28, 256)	0
conv2d_101 (Conv2D)	(None, 28, 28, 512)	1,180,160
batch_normalization_97 (BatchNormalization)	(None, 28, 28, 512)	2,048
conv2d_102 (Conv2D)	(None, 28, 28, 512)	2,359,808
max_pooling2d_8 (MaxPooling2D)	(None, 14, 14, 512)	0
conv2d_103 (Conv2D)	(None, 14, 14, 512)	2,359,808
batch_normalization_98 (BatchNormalization)	(None, 14, 14, 512)	2,048
conv2d_104 (Conv2D)	(None, 14, 14, 512)	2,359,808
max_pooling2d_9 (MaxPooling2D)	(None, 7, 7, 512)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_10 (Dense)	(None, 2048)	51,382,272
dropout_4 (Dropout)	(None, 2048)	0
dense_11 (Dense)	(None, 2048)	4,196,352
dropout_5 (Dropout)	(None, 2048)	0
dense_12 (Dense)	(None, 10)	20,490

Total params: 195,024,096 (743.96 MB)
Trainable params: 65,007,050 (247.98 MB)
Non-trainable params: 2,944 (11.50 KB)
Optimizer params: 130,014,102 (495.96 MB)

CNN Architecture : LeNet-5

Model: "sequential_10"

Layer (type)	Output Shape	Param #
conv2d_121 (Conv2D)	(None, 224, 224, 32)	2,432
batch_normalization_110 (BatchNormalization)	(None, 224, 224, 32)	128
max_pooling2d_20 (MaxPooling2D)	(None, 112, 112, 32)	0
conv2d_122 (Conv2D)	(None, 112, 112, 64)	51,264
batch_normalization_111 (BatchNormalization)	(None, 112, 112, 64)	256
max_pooling2d_21 (MaxPooling2D)	(None, 56, 56, 64)	0
conv2d_123 (Conv2D)	(None, 56, 56, 128)	73,856
batch_normalization_112 (BatchNormalization)	(None, 56, 56, 128)	512
flatten_6 (Flatten)	(None, 401408)	0
dense_23 (Dense)	(None, 256)	102,760,704
dropout_12 (Dropout)	(None, 256)	0
dense_24 (Dense)	(None, 128)	32,896
dropout_13 (Dropout)	(None, 128)	0
dense_25 (Dense)	(None, 10)	1,290

Total params: 102,923,338 (392.62 MB)
Trainable params: 102,922,890 (392.62 MB)
Non-trainable params: 448 (1.75 KB)

CNN Architecture : AlexNet

Model: "sequential_9"

Layer (type)	Output Shape	Param #
conv2d_116 (Conv2D)	(None, 54, 54, 96)	34,944
batch_normalization_105 (BatchNormalization)	(None, 54, 54, 96)	384
max_pooling2d_17 (MaxPooling2D)	(None, 26, 26, 96)	0
conv2d_117 (Conv2D)	(None, 26, 26, 256)	614,656
batch_normalization_106 (BatchNormalization)	(None, 26, 26, 256)	1,024
max_pooling2d_18 (MaxPooling2D)	(None, 12, 12, 256)	0
conv2d_118 (Conv2D)	(None, 12, 12, 384)	885,120
batch_normalization_107 (BatchNormalization)	(None, 12, 12, 384)	1,536
conv2d_119 (Conv2D)	(None, 12, 12, 384)	1,327,488
batch_normalization_108 (BatchNormalization)	(None, 12, 12, 384)	1,536
conv2d_120 (Conv2D)	(None, 12, 12, 256)	884,992
batch_normalization_109 (BatchNormalization)	(None, 12, 12, 256)	1,024
max_pooling2d_19 (MaxPooling2D)	(None, 5, 5, 256)	0
flatten_5 (Flatten)	(None, 6400)	0
dense_20 (Dense)	(None, 4096)	26,218,496
dropout_10 (Dropout)	(None, 4096)	0
dense_21 (Dense)	(None, 4096)	16,781,312
dropout_11 (Dropout)	(None, 4096)	0
dense_22 (Dense)	(None, 10)	40,970

Total params: 140,374,944 (535.49 MB)
Trainable params: 46,790,730 (178.49 MB)
Non-trainable params: 2,752 (10.75 KB)
Optimizer params: 93,581,462 (356.98 MB)

Build the second CNN Model (Transfer Learning)

Using pre-trained models:

- ResNet50
- EfficientNetB0
- InceptionV3

Transfer Learning Models

- **ResNet50**

A deep CNN model with 50 layers, utilizing residual connections to solve the vanishing gradient problem, allowing for efficient training of very deep networks. Pre-trained on ImageNet, it can be fine-tuned for various tasks such as image classification and object detection.

- **EfficientNetB0**

A lightweight, high-performance model with optimized scaling of depth, width, and resolution. EfficientNetB0 is pre-trained on ImageNet and is known for achieving high accuracy with fewer parameters, making it suitable for resource-constrained environments.

- **InceptionV3**

A deep CNN model using Inception modules, which capture multi-scale features through parallel convolution operations with different kernel sizes. Pre-trained on ImageNet, InceptionV3 performs well for large-scale image classification tasks and is efficient at feature extraction.

03

Model Training

Train the first CNN models:

1. CNN-based architecture (simple)
2. CNN Architectures :
   - VGG16
   - LeNet-5
   - AlexNet

Hyperparameters

CNN-based Architecture

Optimizer: SGD (Stochastic Gradient Descent)
Learning rate: 0.0001
Number of Epochs: 10

VGG16

Optimizer: Adam
Learning rate: 0.0001
Number of Epochs: 30
Dropout rate: 0.5

LeNet-5

Optimizer: Adam
Learning rate: 0.001
Number of Epochs: 30
Dropout rate: 0.5

AlexNet

Optimizer: Adam
Learning rate: 0.001
Number of Epochs: 30
Dropout rate: 0.5

Train the second CNN Models (Transfer Learning)

Using pre-trained models:

- ResNet50
- EfficientNetB0
- InceptionV3

Transfer Learning

InceptionV3 & EfficientNetB0 & ResNet50

Optimizer: 'adam'
Number of Epochs: 50
Dropout rate: 0.3

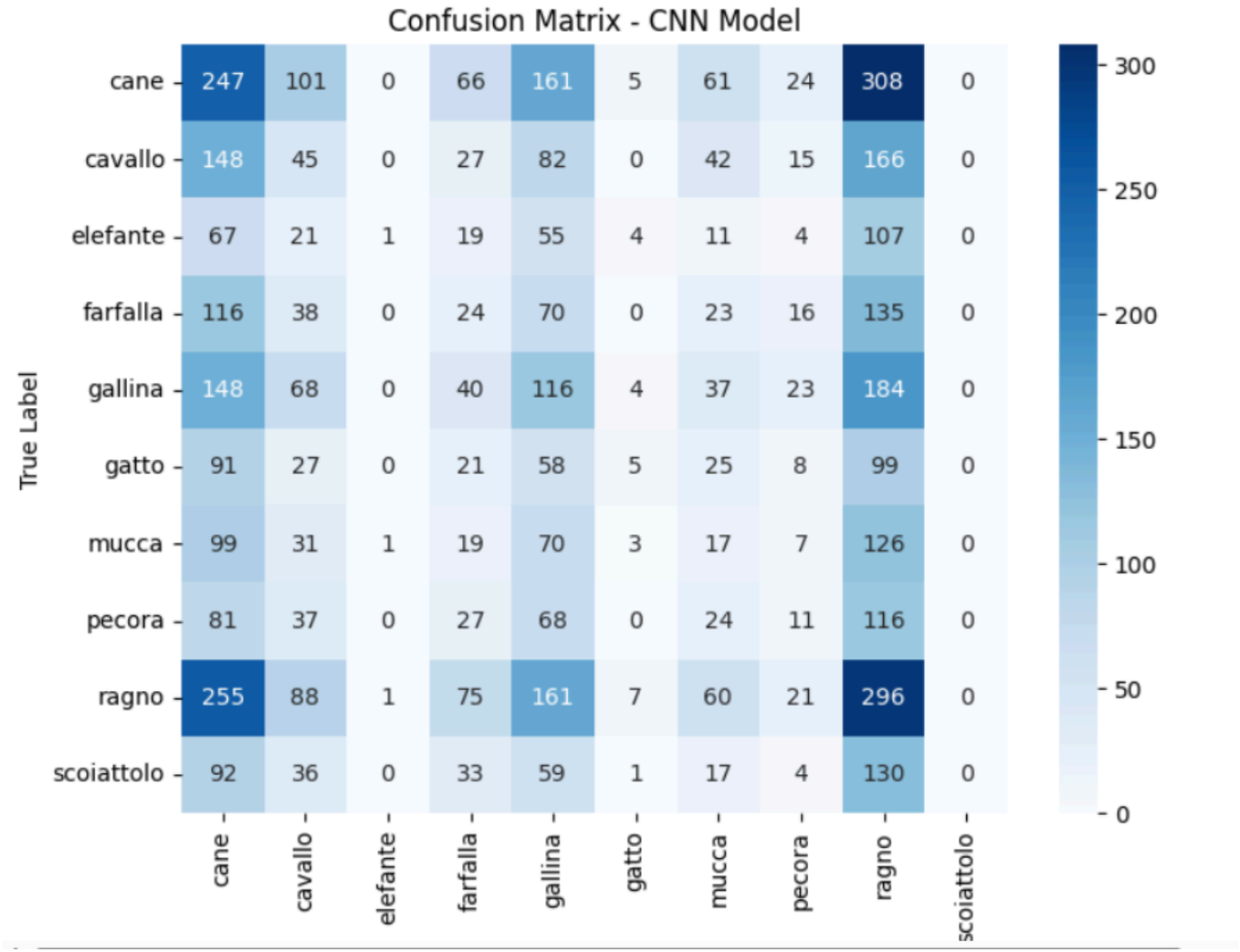
04

Model Evaluation & Selection

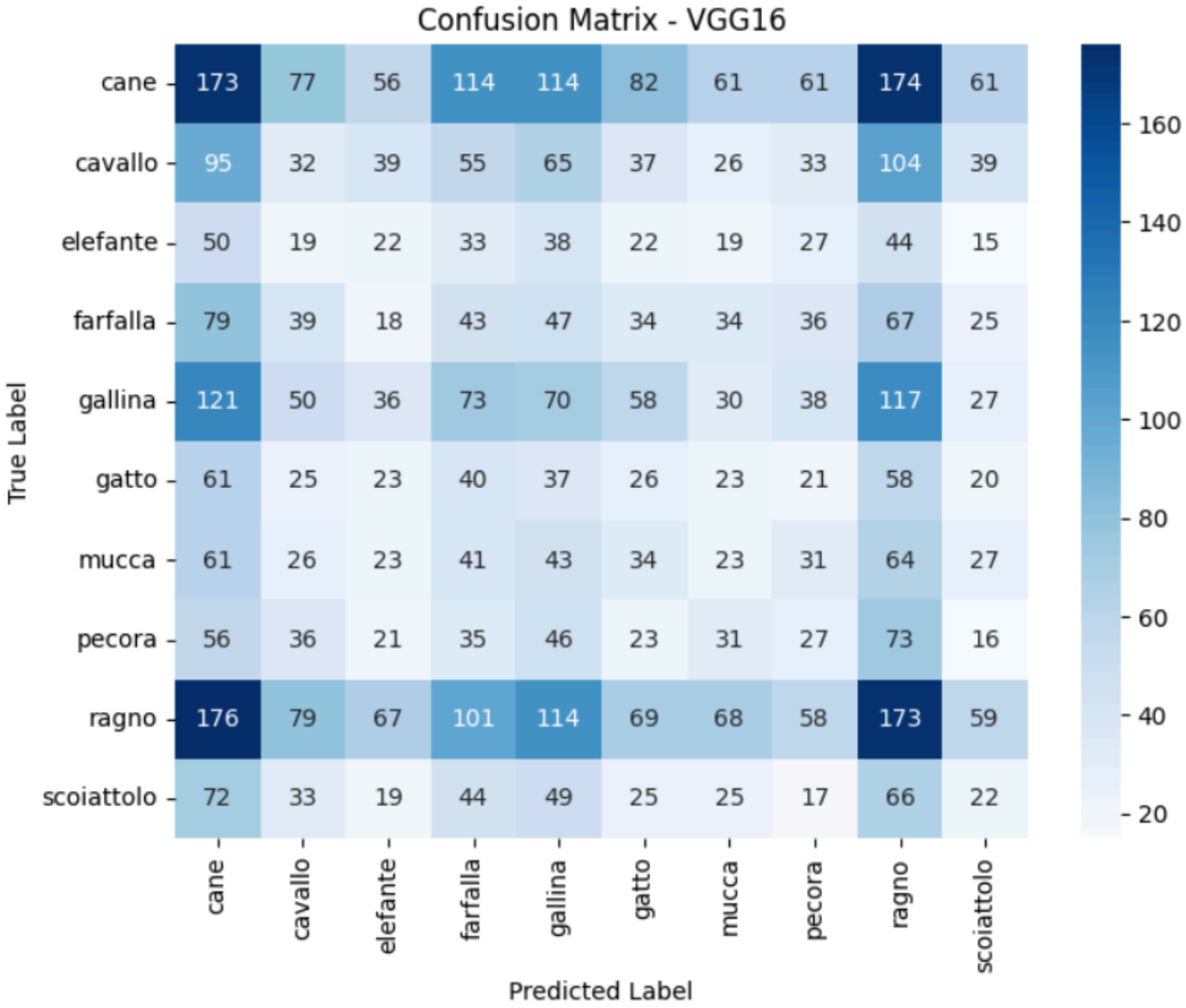
First CNN models:

Metrics	CNN	VGG16	LeNet-5	AlexNet
accuracy	0.1455	0.1167		0.1280
precision	0.1292	0.0978		0.1052
recall	0.0985	0.0982		0.1041
F1-score	0.0869	0.0975		0.1040

Confusion Matrix: CNN

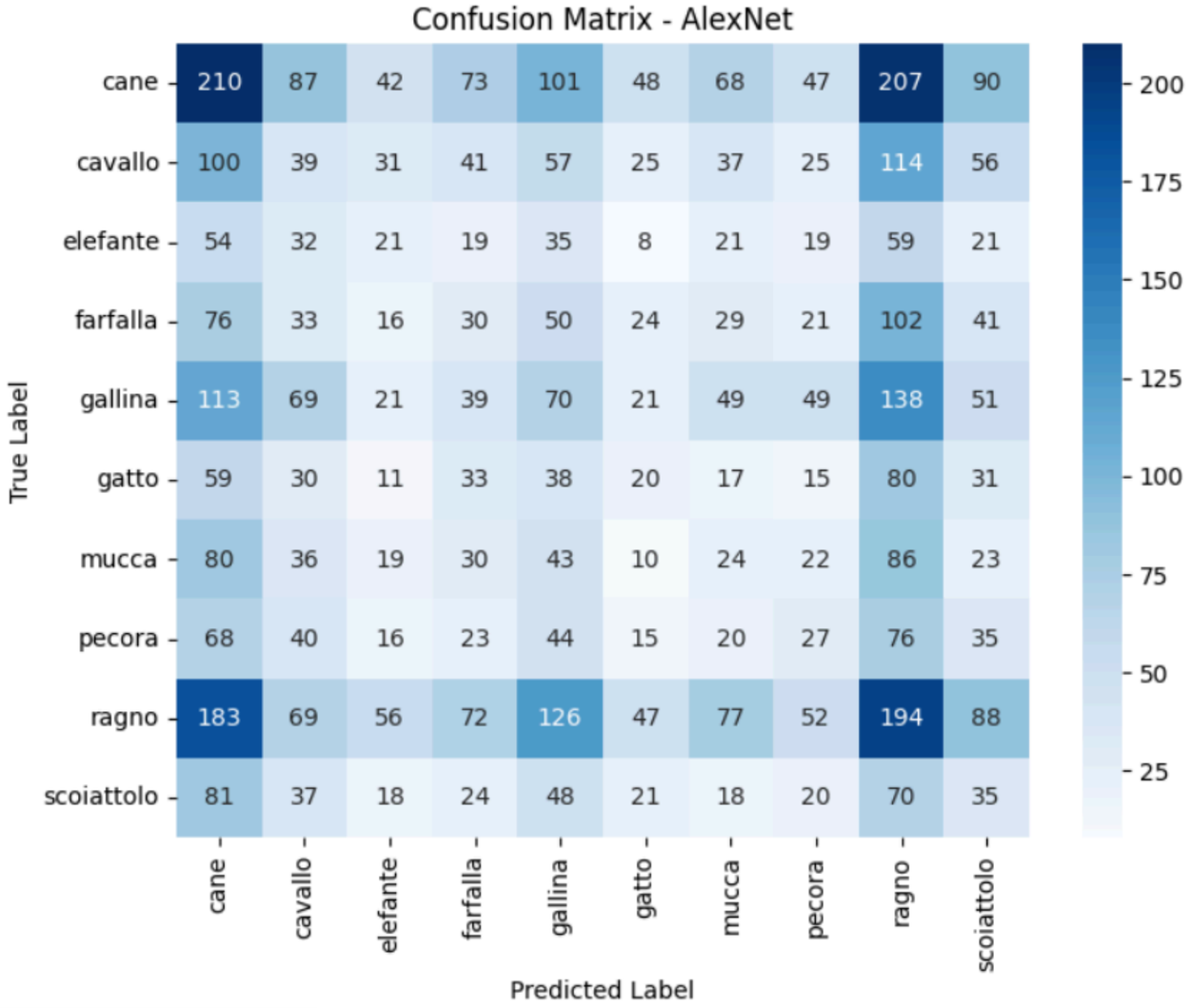


Confusion Matrix: VGG16



Confusion Matrix: LeNet-5

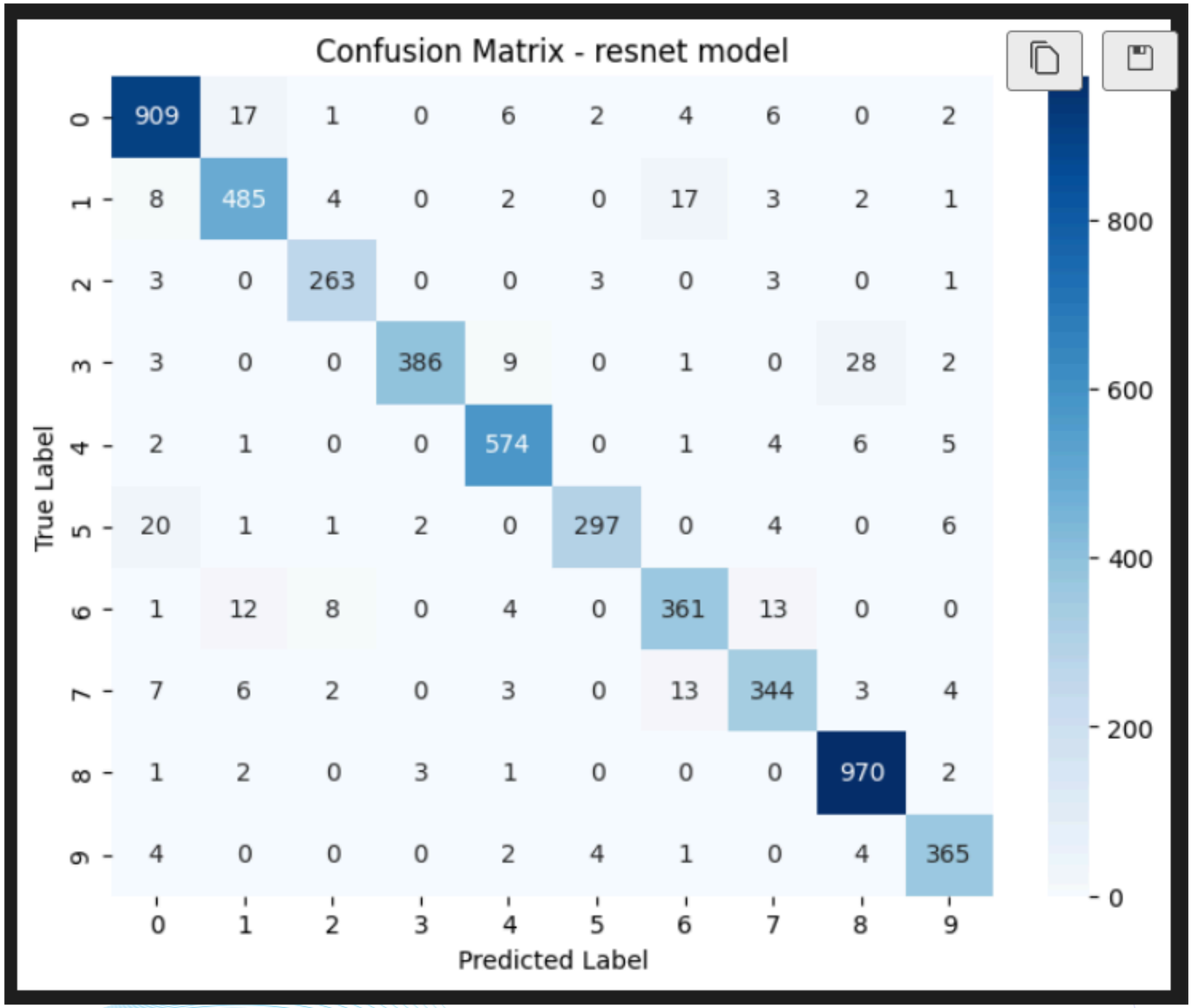
Confusion Matrix: AlexNet



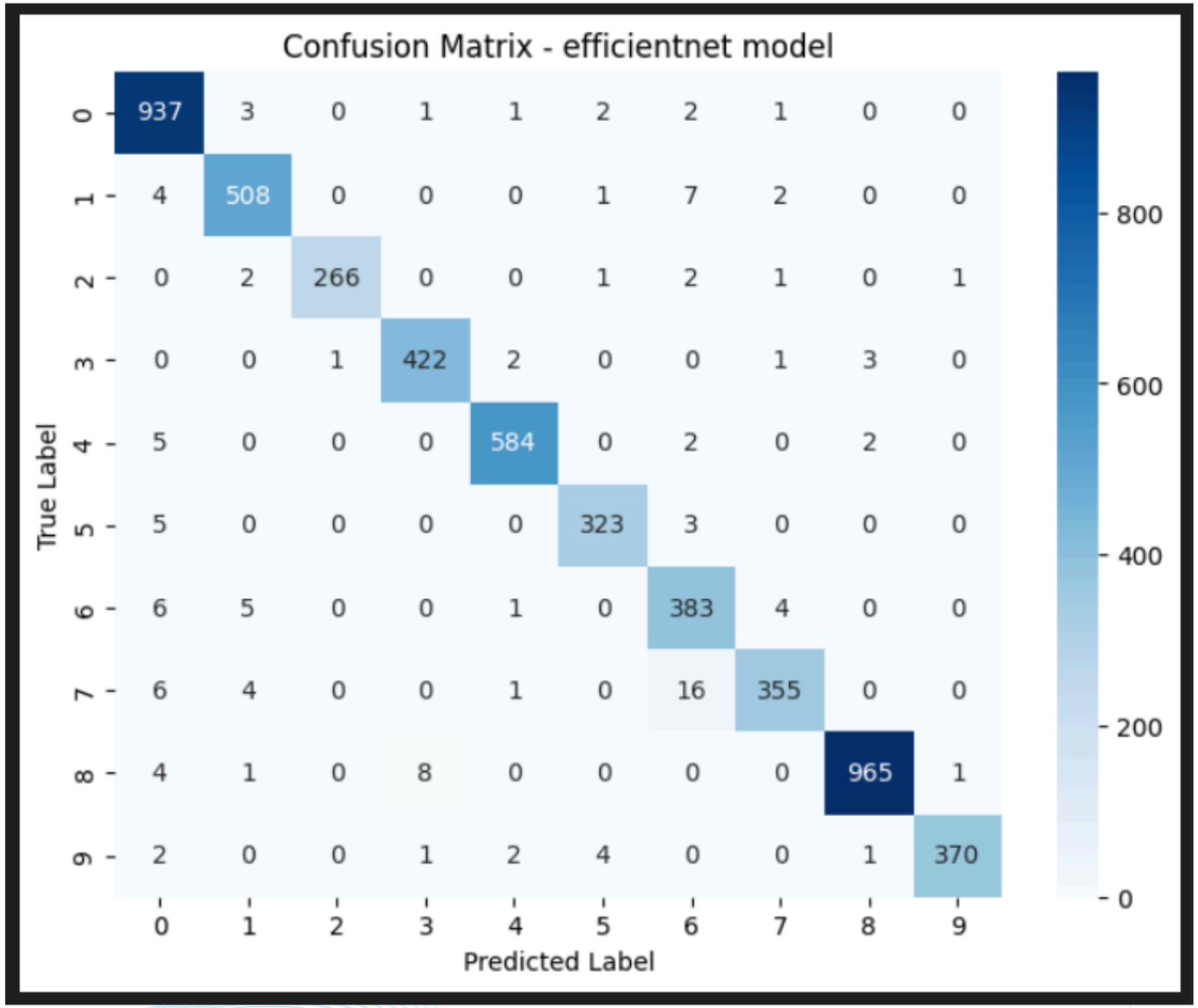
Second CNN model:

Metrics	ResNet50	EfficientNetB0	InceptionV3
accuracy	0.9463	**0.9767**	0.1809
precision	0.9448	**0.9762**	0.0181
recall	0.9374	**0.9730**	0.1000
F1-score	0.9407	**0.9745**	0.0306

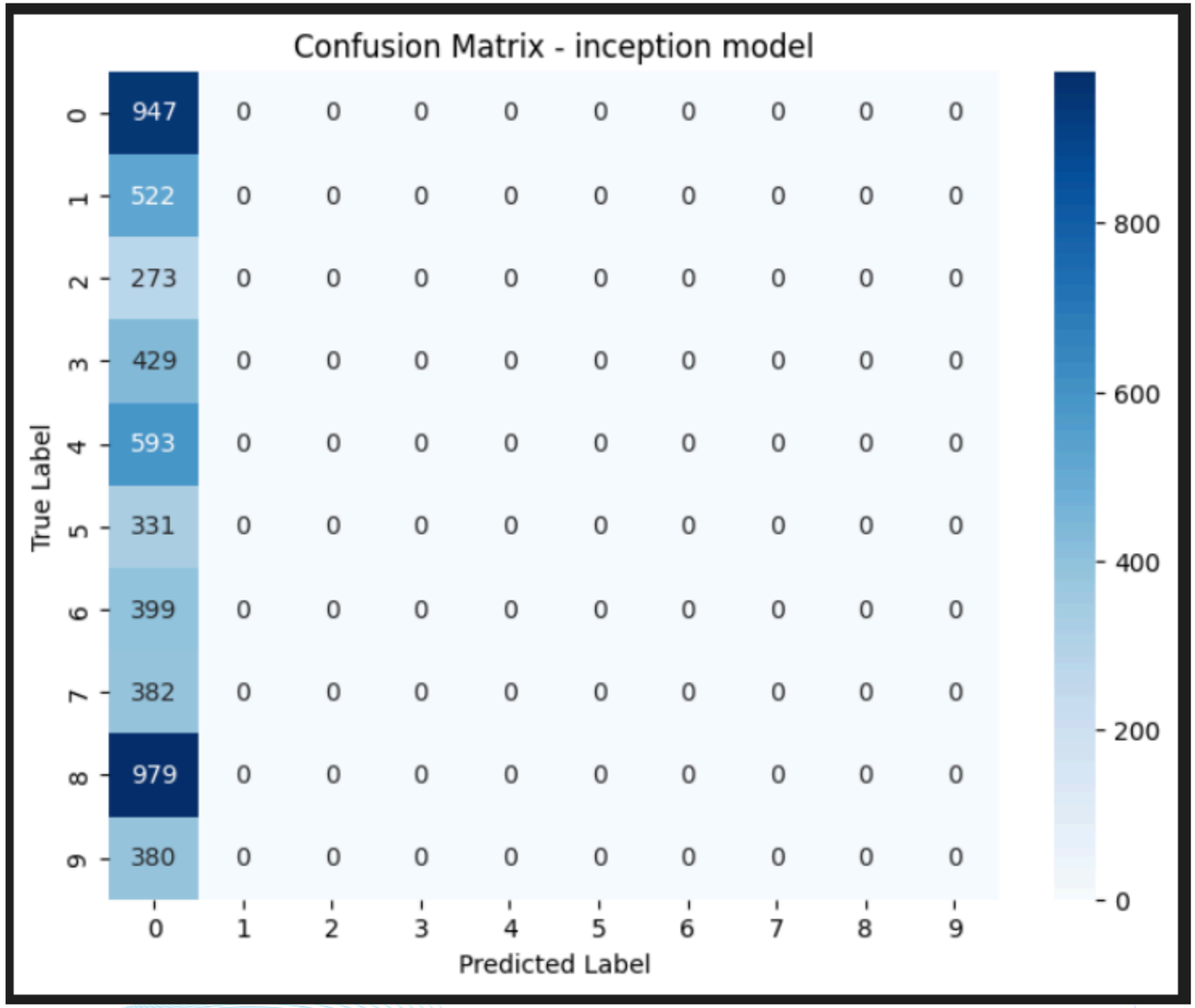
Confusion Matrix: ResNet50



Confusion Matrix: EfficientNetB0



Confusion Matrix: InceptionV3



Between comparisons between the first and second models, the choice fell on

✨**EfficientNetB0**✨

because it is the best-performing model that was built with a transfer learning

05

Conclusions

Conclusions

In conclusion, we covered the key aspects of working with deep learning models for image classification using advanced architectures like CNN, VGG16, LeNet-5, and AlexNet. Through training and evaluation, we tested the performance using various metrics such as accuracy, precision, recall, and F1 score. We also implemented improvements like Dropout and batch normalization to reduce overfitting.

This experience has provided us with a deeper understanding of how to optimize deep learning models and stay up-to-date with advancements in the field of artificial intelligence. Despite the challenges faced in data handling and hyperparameter tuning, the results have been promising, and we are ready for further improvements and experimentation in the future.

Thank you

Do you have any questions?