”Convolutional Neural Network with VGG, ResNet, and Mobilenet”

1st Rene Jerez
Computer Engineering Master Degree
Universidade da Beira Interior
Covilhã, Portugal
rene.jerez@ubi.pt

***Abstract*—Neural networks, a fundamental concept in machine learning courses, are inspired by the human brain to tackle complex, non-linear tasks. They consist of layered neurons that process data through weighted connections and activation functions to capture patterns. Feed-forward networks like multi-layer perceptrons (MLPs) use input, hidden, and output layers to refine predictions. Convolutional Neural Networks (CNNs) excel in computer vision by identifying spatial hierarchies through convolution and pooling layers, enhancing accuracy with activation functions such as ReLU. Learning through backpropagation enables CNNs to adjust weights and minimize errors, making them invaluable in applications like image recognition and diagnostics.**

***Index Terms*—CNN, Neural Network, ReLU, MLP**

I. Introduction

Convolutional Neural Networks (CNNs) are a powerful type of neural network designed to handle visual data effectively. They work by processing images through layers that extract features like shapes and textures while reducing unnecessary information to focus on what matters. Using non-linear activation functions, CNNs can understand complex patterns and relationships, making them ideal for tasks that require recognizing and analyzing visual content.

Over time, CNNs have evolved into highly efficient and accurate models that handle large datasets with ease. Their ability to simplify data while preserving essential details makes them a key tool in artificial intelligence. Whether it's for classification, detection, or other visual tasks, CNNs are central to solving challenging problems in machine learning.

II. Methodology

Our approach involves the AR dataset, consisting of 3,315 RGB images from 136 subjects, was prepared for classification by resizing images to 224x224 pixels, normalizing pixel values to [0, 1], and applying data augmentation (e.g., rotation, flipping, scaling) to enhance diversity and prevent overfitting. The dataset was split into training (70), validation (15), and testing (15) subsets, ensuring class balance.

The study focused on four classification tasks: predicting subject ID, facial expression, gender, and the presence of glasses. Models were implemented using Python and Keras, employing both learning-from-scratch and transfer learning approaches with architectures such as VGG, ResNet, and Inception.

Training used categorical or binary cross-entropy loss functions, the Adam optimizer, and early stopping to avoid overfitting. Performance metrics, including accuracy, precision, recall, and F1-score, were used to evaluate models. A comparative analysis was conducted to measure the effectiveness of each architecture and approach across the tasks, considering both accuracy and training efficiency.

A. Model Implementation

The construction of this model sets up a workflow to train and test different deep learning models for classifying images. It starts by prepping the data: grayscale images are reshaped and resized to fit the input size required by the chosen models (ResNet, VGG, or MobileNet). These images are then converted to RGB, normalized (scaled to values between 0 and 1), and the labels are converted into one-hot encoded formats for multi-class classification. The data is split into training and test sets, with 80 percent for training and 20 percent for testing.

The script supports three types of models: ResNet, VGG, and MobileNet. Each model uses a pre-trained base, if available, followed by a few added layers: global average pooling, a dense layer with 256 units, and a softmax output layer to handle the classifications. The models are compiled with the Adam optimizer and a loss function that adjusts automatically based on the number of classes. Once set up, the models are trained on the data for a specified number of epochs and batch size.

After training, the code evaluates the model's accuracy on the test set and prints the results. It also includes an example to process a CSV file, clean up the data, and prepare it for training. Finally, the script trains ResNet, VGG, and MobileNet models using this setup, so you can compare how well each performs on the same task. It's a straightforward way to explore and test different architectures on your dataset.

The models took quite a bit of time to run. ResNet training lasted more than 2 hours, VGG took about 3 hours, and MobileNet finished in less than 2 hours. When using ImageNet weights, the training became even slower, with each epoch taking significantly more time to complete. Due to this extended runtime, the process had to be stopped before finishing.

Identify applicable funding agency here. If none, delete this.

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dense (Dense)	(None, 256)	524,544
dense_1 (Dense)	(None, 2)	514

Fig. 1. Resnet Details

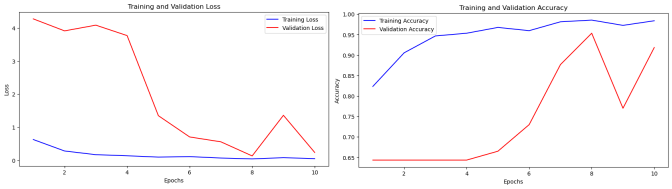

Fig. 2. Resnet Metrics

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 1, 1, 512)	14,714,688
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 512)	0
dense_2 (Dense)	(None, 256)	131,328
dense_3 (Dense)	(None, 2)	514

Fig. 3. VGG Details

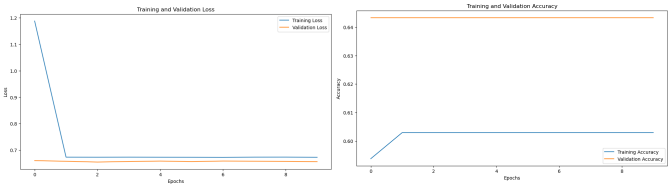

Fig. 4. VGG Metrics

Layer (type)	Output Shape	Param #
mobilenet_1.00_96 (Functional)	(None, 3, 3, 1024)	3,228,864
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1024)	0
dense (Dense)	(None, 128)	131,200
dense_1 (Dense)	(None, 2)	258

Fig. 5. Mobilenet Details

III. Conclusion

For the conclusion, I'm still running models for ID, category, and expression, but the script is designed to handle any target in the AR dataset. I used the PCA script provided by the professor to reduce the data and created a specific model for each target. To save time, I avoided using ImageNet weights and saved all trained models as '.h5' files for future use.