Lab Report — Using CNN Techniques in Python (Keras/TensorFlow)

1. Lab Objective

The objective of this lab is to become familiar with Convolutional Neural Networks (CNNs) and their implementation in Python for **image classification** using Keras/TensorFlow. The work includes: (i) preparing and loading an image dataset, (ii) training a CNN model, (iii) analyzing the effect of data augmentation, (iv) proposing an alternative CNN architecture with regularization, and (v) adapting the pipeline to **grayscale (luminance-only)** classification.

2. Computing Environment

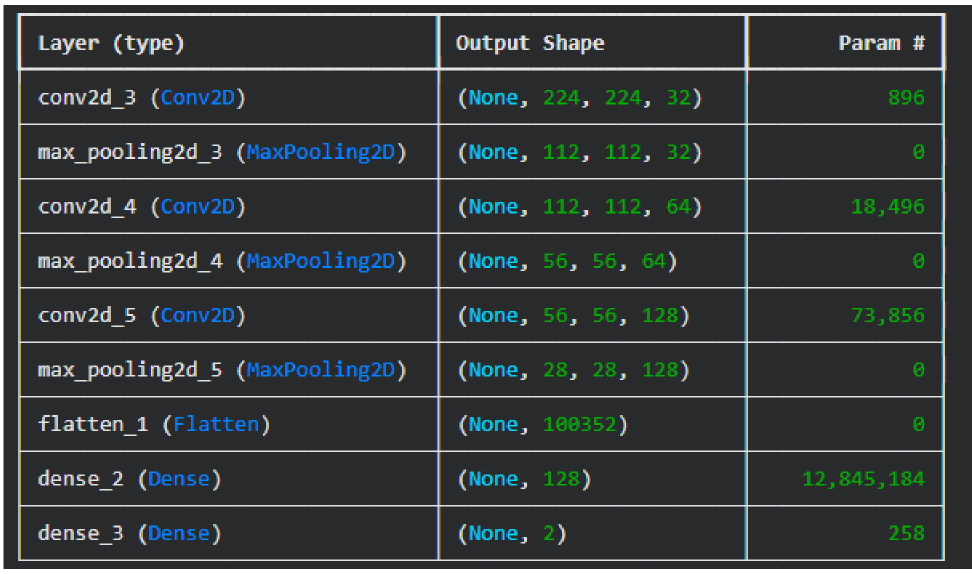
- **Language:** Python
- **Framework:** TensorFlow / Keras
- **Data pipeline:** ImageDataGenerator + flow_from_directory
- **Visualization:** Matplotlib
- **Input size:** 224 × 224
- **Batch size:** 16
- **Epochs:** 10

3. Dataset and Folder Structure

The dataset is provided as a **ZIP** archive and must be extracted before training. The directory structure required by Keras is:

- train/
    - class_1/
    - class_2/
- test/ (or validation/)
    - class_1/
    - class_2/

This structure allows Keras to automatically infer labels from folder names and perform supervised learning.



Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 224, 224, 32)	896
max_pooling2d_3 (MaxPooling2D)	(None, 112, 112, 32)	0
conv2d_4 (Conv2D)	(None, 112, 112, 64)	18,496
max_pooling2d_4 (MaxPooling2D)	(None, 56, 56, 64)	0
conv2d_5 (Conv2D)	(None, 56, 56, 128)	73,856
max_pooling2d_5 (MaxPooling2D)	(None, 28, 28, 128)	0
flatten_1 (Flatten)	(None, 100352)	0
dense_2 (Dense)	(None, 128)	12,845,184
dense_3 (Dense)	(None, 2)	258

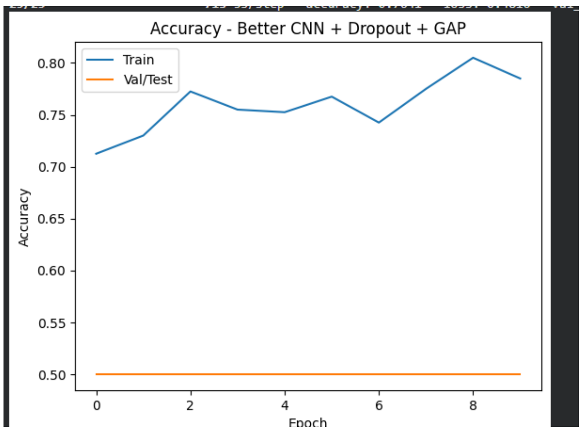
4. General Methodology

The experimental workflow followed these steps:

1. **Extract** the ZIP dataset.

2. **Load and preprocess** images using ImageDataGenerator.
3. **Train** multiple CNN models:
    - Baseline CNN (Flatten + Dense)
    - Improved CNN (BatchNorm + Dropout + GlobalAveragePooling)
    - Grayscale CNN (luminance-only)
4. **Evaluate** performance using accuracy and loss on training and validation/test sets.
5. **Analyze** the learning curves to detect overfitting/underfitting and generalization quality.

5. Model 1 — Baseline CNN (Flatten + Dense)



5.1 Architecture

The baseline model uses a standard CNN pipeline:

- 3 blocks: **Conv2D + MaxPooling2D**
- Flatten()
- Dense(128)
- Dense(2) with Softmax

5.2 Model Complexity (Parameters)

The model summary reports approximately **12,938,690 trainable parameters**. This high number is mainly caused by the Flatten() layer, which creates a very large vector before the Dense layer, producing a large number of weights.

5.3 Observed Results

From the training logs and plots:

- **Training accuracy** increases up to ~**1.00 (100%)**
- **Validation/Test accuracy** remains around **0.81–0.87** (≈ **0.85**)
- **Validation loss** tends to increase near the end, even while accuracy remains fairly high

5.4 Interpretation (Overfitting)

This behavior indicates **overfitting**:

- The model learns the training set almost perfectly (memorization).
- Generalization to validation/test is limited (≈ 85%).
- The increasing validation loss suggests the model becomes over-confident on wrong predictions (poor calibration), which is a common symptom when the model is too complex for the dataset size.

6. Exercise 3 — Effect of Removing Data Augmentation

6.1 Experiment Description

The lab requires replacing the augmentation pipeline:

With augmentation (original TP setup):

- rescale=1./255
- shear_range=0.2
- zoom_range=0.2
- horizontal_flip=True

Without augmentation (modified setup):

- rescale=1./255 only

6.2 Expected Effect on Classification Performance

- **With augmentation:**
  - increases variability in training samples (virtual data expansion)
  - improves generalization and reduces overfitting
  - may slow training convergence but often stabilizes validation performance
- **Without augmentation:**
  - training images remain limited in variety
  - the model may overfit faster
  - validation performance becomes less stable and may degrade on real unseen samples

6.3 Conclusion (Exercise 3)

Given that overfitting is already visible in the baseline model, **data augmentation is strongly recommended** to improve generalization.

7. Exercise 4 — Alternative CNN Architecture (Dropout + GAP)

7.1 Purpose

To define a different CNN architecture including:

- multiple convolutional layers,
- **Dropout** regularization,
- and replacing Flatten() with **GlobalAveragePooling2D (GAP)** to drastically reduce parameter count.

7.2 Proposed Architecture

- Conv2D(32) + BatchNorm + ReLU + MaxPooling
- Conv2D(64) + BatchNorm + ReLU + MaxPooling
- Conv2D(128) + BatchNorm + ReLU + MaxPooling
- **GlobalAveragePooling2D**
- Dropout(0.5)
- Dense(128) + Dropout(0.3)
- Dense(2) + Softmax

7.3 Complexity

The model summary reports approximately **110,914 parameters**, which is a major reduction compared to the baseline (≈ 12.9M). This typically reduces the risk of overfitting.

7.4 Observed Results (From Your Plots)

The “Better CNN + Dropout + GAP” results show:

- training accuracy increases toward ~0.78–0.80
- **validation accuracy stays around 0.50** (flat line)
- validation loss increases substantially (≈ 1.5–1.9)

7.5 Analysis: Why is Validation Accuracy Stuck at 0.50?

A constant **0.50 accuracy** in a binary classification task often indicates a **pipeline/configuration issue**, not only a model issue. Common causes include:

1. **Label mapping mismatch between train and validation generators**
    - Must verify that:
      train_generator.class_indices == val_generator.class_indices
2. **Mismatch between output layer / loss / class_mode**
    - If class_mode="categorical" → use 2-neuron Softmax + categorical_crossentropy
    - If class_mode="binary" → use 1-neuron Sigmoid + binary_crossentropy
      Any mismatch can break validation learning.
3. **Validation folder structure problem**
    - Missing class folder, different folder names, or wrong paths.
4. **Class imbalance (less likely to create a perfect flat 0.50, but still possible)**
    - Requires checking sample counts per class.

7.6 Conclusion (Exercise 4)

- The architecture is **academically correct and well-regularized**.
- However, the validation behavior suggests the need to **verify data/labels and generator settings** before concluding about model quality.

8. Exercise 5 — Grayscale (Luminance-Only) Classification

8.1 Required Modifications

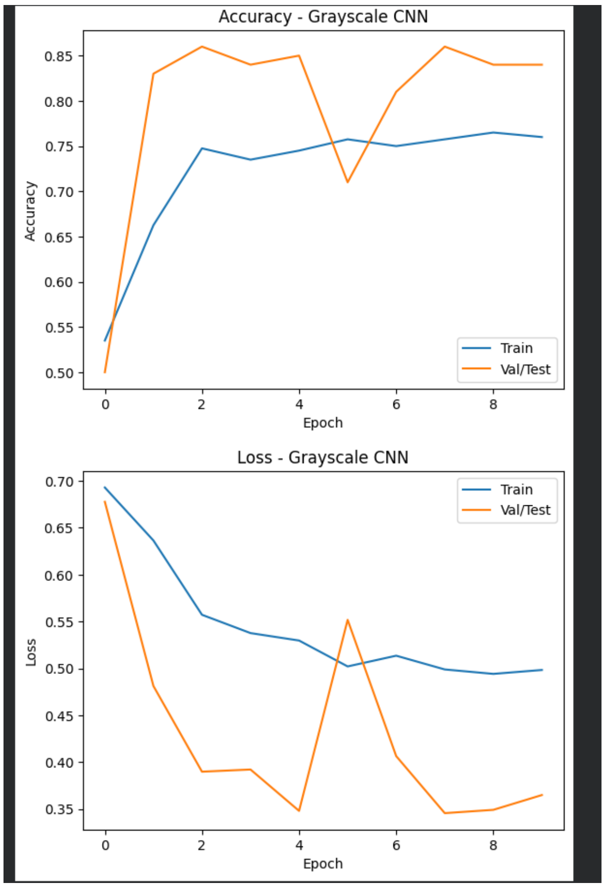
To classify using luminance only:

- load images with color_mode="grayscale"
- set model input shape to (224, 224, 1)
- keep normalization/augmentation as needed

8.2 Observed Results (From Your Grayscale Curves)

- validation/test accuracy reaches approximately **0.84–0.86**
- training accuracy stabilizes around **0.75–0.77**
- validation loss remains relatively low (~0.35–0.40), with minor fluctuations

8.3 Interpretation



These results indicate that grayscale information (shape/texture) is sufficient for good classification performance in this dataset. The fact that validation accuracy can appear higher than training accuracy can be explained by stronger augmentation on training data (training becomes "harder") or statistical variation due to dataset size.

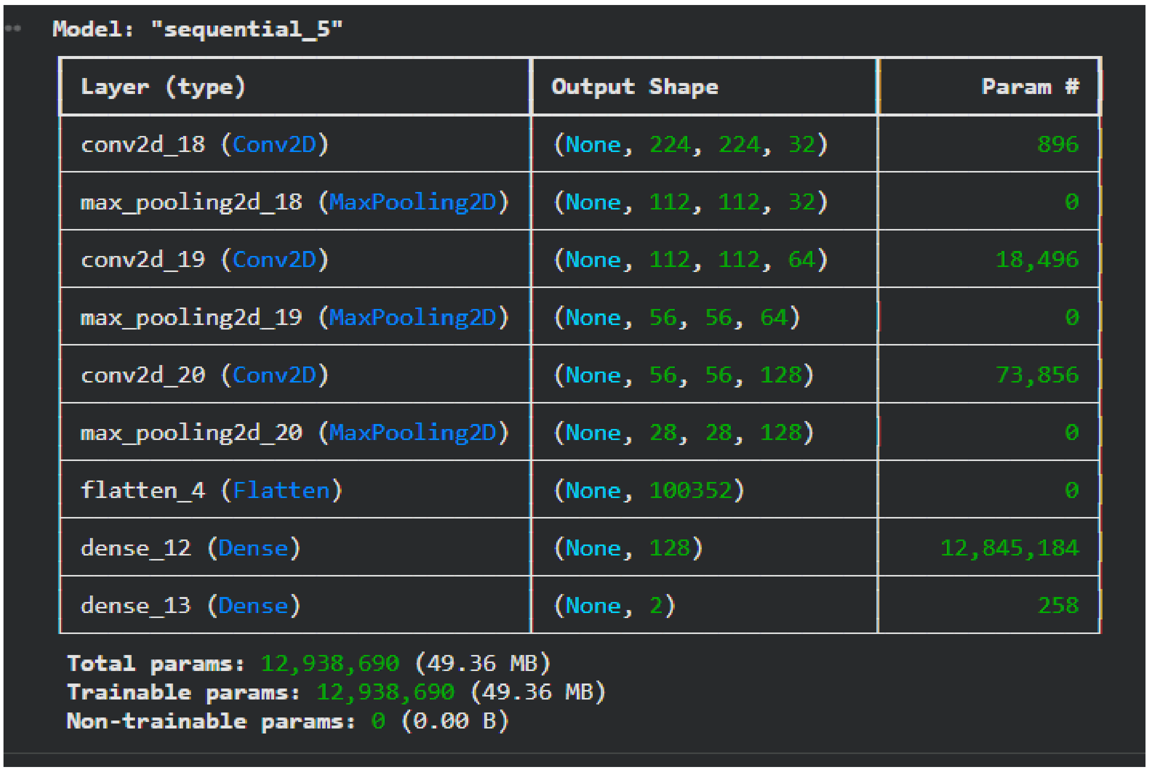
9. Comparative Summary (Key Findings)

- **Baseline CNN:** good validation accuracy (~0.85) but clear **overfitting** due to very high parameter count (Flatten + Dense).
- **Improved CNN (Dropout + GAP):** theoretically better generalization due to fewer parameters, but validation stuck at **0.50** indicates a likely **label/generator issue**.
- **Grayscale CNN:** stable and good validation (~0.84–0.86), showing the task can be solved using luminance-only features.

10. Technical Recommendations (Professional Practice)

To finalize the lab robustly, the following checks are recommended:

1. Print and compare:
    - train_generator.class_indices
    - val_generator.class_indices
2. Verify that validation/test folder contains **exactly the same class subfolders** as training.
3. Add evaluation metrics:
    - confusion matrix, precision/recall/F1-score
4. Use callbacks:
    - EarlyStopping to prevent overfitting in baseline models
5. For small datasets, consider **transfer learning** (MobileNetV2, VGG16, ResNet) as an extension.



```
Model: "sequential_5"
```

Layer (type)	Output Shape	Param #
conv2d_18 (Conv2D)	(None, 224, 224, 32)	896
max_pooling2d_18 (MaxPooling2D)	(None, 112, 112, 32)	0
conv2d_19 (Conv2D)	(None, 112, 112, 64)	18,496
max_pooling2d_19 (MaxPooling2D)	(None, 56, 56, 64)	0
conv2d_20 (Conv2D)	(None, 56, 56, 128)	73,856
max_pooling2d_20 (MaxPooling2D)	(None, 28, 28, 128)	0
flatten_4 (Flatten)	(None, 100352)	0
dense_12 (Dense)	(None, 128)	12,845,184
dense_13 (Dense)	(None, 2)	258

```
Total params: 12,938,690 (49.36 MB)
Trainable params: 12,938,690 (49.36 MB)
Non-trainable params: 0 (0.00 B)
```

Conclusion

This lab demonstrates the full CNN pipeline for image classification: dataset preparation, preprocessing, model training, and evaluation. The baseline model achieves good validation accuracy but suffers from overfitting due to its high parameter count. A regularized architecture using Dropout and GlobalAveragePooling is a strong theoretical improvement, but the observed validation behavior (accuracy fixed at 0.50) suggests a configuration issue in the validation pipeline. Finally, grayscale classification produces stable performance, indicating that luminance-based features are sufficient for the considered classes.

Realised by roukbi mohammed elseddik