# Product Picture Classification

## CV Project

### Repo Link

https://github.com/salmamuhammede/Product-Picture-Classification-CNN-model.git

25/06/2024

# PROJECT IDEA

The project involves a Clothing Classifier using a CNN built with TensorFlow/Keras. It processes images of T-shirts, Trousers, Shoes, and Glasses, training a model to classify these items. The workflow includes loading images from directories, resizing them to a uniform size, normalizing pixel values, and splitting data into training, validation, and test sets. A CNN is trained on this data, and its performance is evaluated through accuracy metrics, confusion matrices, and classification reports. The model's predictions are visualized by showing images from the test set with their predicted labels, making it a practical tool for automatically categorizing clothing images.

# LIBRARIES

TensorFlow/Keras:

- Usage: TensorFlow/Keras is utilized to construct and train the Convolutional Neural Network (CNN) for image classification.

OpenCV (cv2):

- Usage: OpenCV is used for image processing tasks such as reading images from directories (cv2.imread), resizing images (cv2.resize), and converting images between different color spaces (cv2.cvtColor). It plays a key role in preprocessing the images before feeding them into the CNN.

NumPy:

- Usage: NumPy supports numerical operations on arrays, which is essential for image data manipulation.

Matplotlib:

- Usage: Matplotlib is used for visualizing the images and results.

Seaborn:

Scikit-learn:

- Usage: Scikit-learn provides utilities for evaluating the model, such as generating the confusion matrix (confusion_matrix) and classification reports (classification_report).

OS:

- Usage: The OS module is used for directory and file handling operations.

# METHODOLOGY

1. Data Preparation

1.1 Load and Preprocess Images:

- Objective: Read and preprocess images from directories to prepare them for model training.
- Steps:
  - Define a function loadAndPreprocessImages that:
    - Iterates through image directories.
    - Reads images using OpenCV (cv2.imread).
    - Resizes images to a fixed size of 128x128 pixels using cv2.resize.
    - Normalizes pixel values by dividing by 255.0.
    - Appends images and corresponding labels to lists.
  - Convert lists to NumPy arrays for efficient handling and return them.

1.2 Split the Dataset:

- Objective: Divide the dataset into training, validation, and test sets.
- Steps:
  - Load and preprocess images from train_dir, val_dir, and test_dir using loadAndPreprocessImages.
  - Assign preprocessed images and labels to X_train, y_train, X_val, y_val, X_test, and y_test.

# METHODOLOGY

2. Model Development

2.1 Define the CNN Model:

- Objective: Build a Convolutional Neural Network (CNN) to classify images into categories.
- Steps:
  - Use the Sequential API from TensorFlow/Keras to stack layers.
  - Add convolutional layers (Conv2D) with decreasing filters: 128, 64, and 32.
  - Use ReLU activation functions in convolutional layers.
  - Include max-pooling layers (MaxPooling2D) after each convolutional layer to reduce spatial dimensions.
  - Flatten the output from the last convolutional layer.
  - Add a fully connected (dense) layer with 128 units and ReLU activation.
  - Include a dropout layer (Dropout) with a rate of 0.5 to reduce overfitting.
  - Add an output layer with 4 units (for the 4 categories) and a softmax activation function.

2.2 Compile the Model:

- Objective: Configure the learning process of the model.
- Steps:
  - Use the Adam optimizer with a learning rate of 0.001.
  - Define the loss function as sparse_categorical_crossentropy since labels are integers.
  - Specify evaluation metric as accuracy.

# METHODOLOGY

3. Model Training

3.1 Train the Model:

- Objective: Fit the model to the training data and validate on validation data.
- Steps:
  - Train the model using model.fit with X_train and y_train.
  - Set the number of epochs to 20 and batch size to 32.
  - Provide validation data (X_val, y_val) for performance monitoring.
  - Use callbacks for early stopping and model checkpointing to save the best model and prevent overfitting.

4. Model Evaluation

4.1 Evaluate the Model:

- Objective: Assess the model's performance on unseen test data.
- Steps:
  - Evaluate the model on the test set (X_test, y_test) using model.evaluate.
  - Print the test accuracy.

4.2 Generate Predictions:

- Objective: Obtain the model's predictions on the test set.
- Steps:
  - Use model.predict on X_test to get class probabilities.
  - Convert probabilities to class labels using np.argmax.

# METHODOLOGY

5. Model Analysis

5.1 Create Confusion Matrix:

- Objective: Visualize the model's performance across different classes.
- Steps:
  - Compute the confusion matrix using confusion_matrix from scikit-learn.
  - Plot the confusion matrix using Seaborn's heatmap (sns.heatmap) for clarity.

5.2 Generate Classification Report:

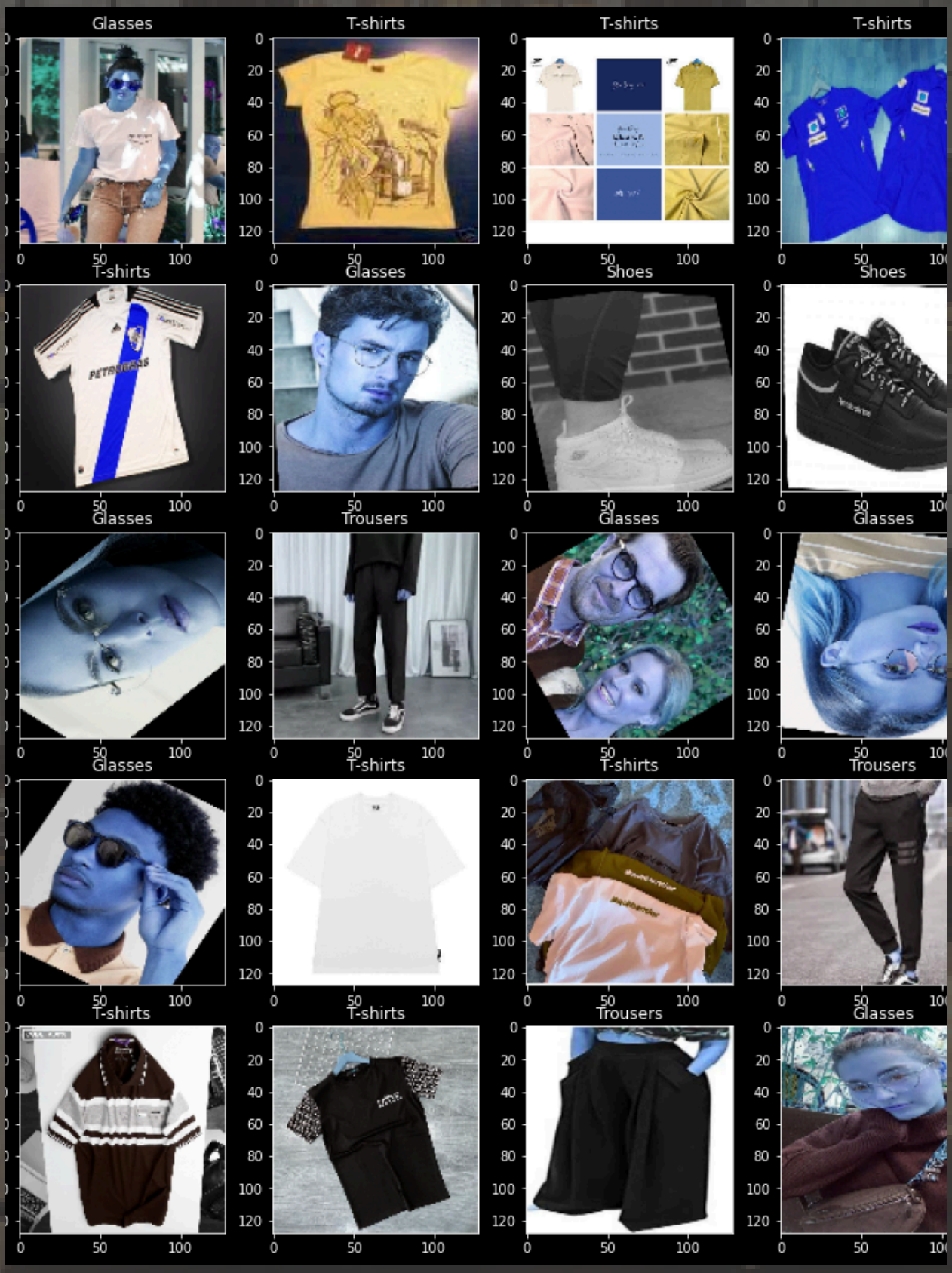
- Objective: Provide a detailed performance summary including precision, recall, and F1-score.
- Steps:
  - Use classification_report from scikit-learn to generate the report.
  - Print the report to analyze the performance across all classes.

6. Visualization

6.1 Visualize Sample Images:

- Objective: Display random images from the training set to verify data loading and preprocessing.
- Steps:
  - Plot a grid of random images from X_train using Matplotlib.
  - Use plt.imshow to display images and plt.title to show their respective categories.

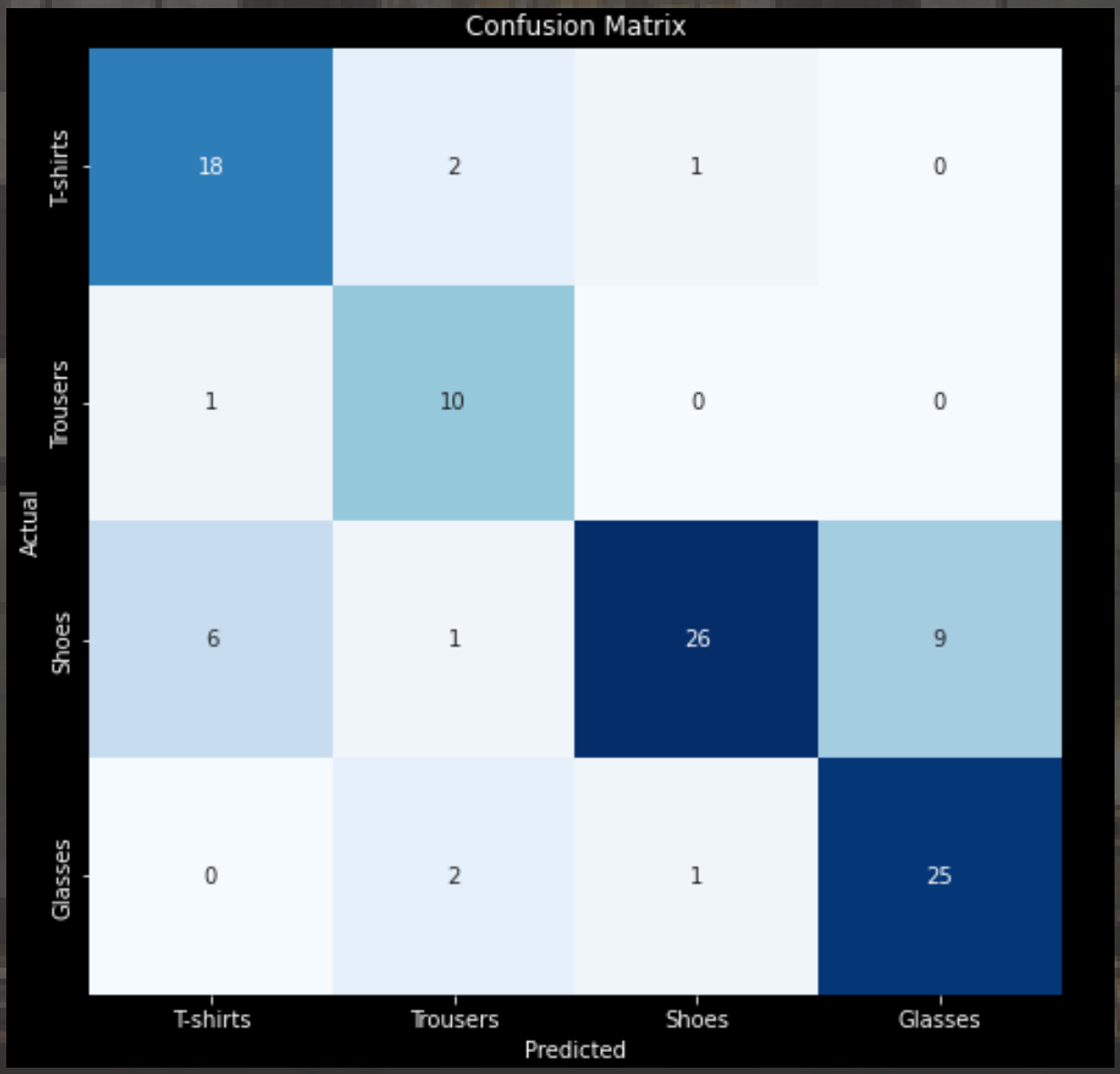
# RESULT SNAPSHOTS
# (RANDOM TRAINING SAMLPE)



# RESULT SNAPSHOTS (MODEL SUMMARY)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 128)     3584

 max_pooling2d (MaxPooling2  (None, 63, 63, 128)       0
 D)

 conv2d_1 (Conv2D)           (None, 61, 61, 64)        73792

 max_pooling2d_1 (MaxPoolin  (None, 30, 30, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 28, 28, 32)        18464

 max_pooling2d_2 (MaxPoolin  (None, 14, 14, 32)        0
 g2D)

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 128)               802944

 dropout (Dropout)           (None, 128)               0

...
Trainable params: 899300 (3.43 MB)
Non-trainable params: 0 (0.00 Byte)
```

# RESULT SNAPSHOTS (CONFUSSION MATRIX)




The confusion matrix reveals that most predictions fall correctly along the diagonal, indicating a high accuracy rate overall. However, the class for "Glasses" shows a higher proportion of incorrect predictions compared to other classes. This discrepancy could be attributed to several factors, including the quality and quantity of the dataset, as well as the model's parameters and architecture.

# RESULT SNAPSHOTS
## (EVALUTION REPORT)

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| T-shirts | 0.72 | 0.86 | 0.78 | 21 |
| Trousers | 0.67 | 0.91 | 0.77 | 11 |
| Shoes | 0.93 | 0.62 | 0.74 | 42 |
| Glasses | 0.74 | 0.89 | 0.81 | 28 |
| accuracy | | | 0.77 | 102 |
| macro avg | 0.76 | 0.82 | 0.78 | 102 |
| weighted avg | 0.80 | 0.77 | 0.77 | 102 |

- Overall Accuracy: Achieving 77% accuracy is a solid starting point, indicating the model correctly classifies the majority of instances.
- High Precision in Certain Classes: For example, the precision for Shoes is very high at 0.93, showing the model is often right when it predicts this class.
- Good Recall in Some Classes: The recall for T-shirts and Glasses is high (0.86 and 0.89, respectively), meaning the model detects most instances of these classes correctly.

# RESULT SNAPSHOTS
## (RANDOM TEST SAMPLE)

# CHALLENGES

- Data Collection and Preparation: Initially, finding a unified dataset containing separate photos for the four categories (T-shirts, Trousers, Shoes, Glasses) was challenging. To address this, I combined data from four different datasets and meticulously organized them into three distinct folders: train, validation, and test. This strategy helped overcome the problem of mixed category datasets and facilitated better model training and evaluation.

- CNN Model Structure and Debugging: Transitioning from using Faster R-CNN and YOLO to a standard CNN for this task presented several challenges. The process involved extensive debugging, which was time-consuming and resource-intensive, straining my PC's CPU and storage capabilities. Each debugging session often required hours and multiple reboots. I experimented with different epochs, image sizes (128 and 256), and batch sizes to optimize performance. Despite the complexity and difficulties, achieving a final accuracy of 0.77 from a peak of 0.8 was a rewarding experience. This project not only enhanced my understanding of CNNs but also provided valuable learning opportunities despite the many challenges faced along the way.