**National University of Computer and Emerging Sciences, Karachi  
FAST School of Computing**Logo

Description automatically generated

**Deep Learning and Perception, Spring 2024**

**Project Report**

**Currency Detection**

**Group Members**

**Ahzam Imam 20K-1612**

**Aaliyan Khan 20K-1856**

**Reyan 20K-0194**

**Currency Detection Model**

This report provides a comprehensive analysis of our currency detection model built using TensorFlow and Keras. It covers various aspects, including code breakdown, data preparation, model architecture, training process, evaluation, and deployment.

**Code Breakdown**

* **Imports:**
  + Standard libraries like os, numpy, matplotlib.pyplot are imported for general functionalities.
  + Keras/TensorFlow libraries are imported for deep learning tasks:
    - ImageDataGenerator for image augmentation during training.
    - image for image loading and preprocessing.
    - Layers and optimizers for building the model.
* **Data Preparation:**
  + **Target Size:** We defined target\_size=(256, 256) for resizing images during training and testing.
  + **Train and Validation Data Generators:**
    - ImageDataGenerator is used to create training and validation data generators.
    - Augmentation techniques like rescaling, rotation, shifting, zooming, and flipping are applied to the training data for better generalization.
    - Validation split of 0.2 is set within the training data generator.
  + **Flow from Directory:**
    - train\_datagen.flow\_from\_directory is used to create training and validation data generators from the "dataset/train" and "dataset/test" directories, respectively.
    - These directories should be structured with subfolders for each currency class.
    - Batch size of 64 and categorical mode are specified for training data.
  + **Class Indices:**
    - train\_generator.class\_indices is used to retrieve a dictionary mapping class labels to numerical indices used by the model.
* **Model Architecture:**
  + **Sequential Model:**
    - A sequential model is used, where layers are added one after another.
  + **Convolutional Layers:**
    - Three convolutional layers are used with:
      * Filter sizes of 3x3 kernels.
      * ReLU activation function for non-linearity.
      * Max pooling layers (pool\_size=(2, 2)) for dimensionality reduction and capturing spatial features.
  + **Flatten Layer:**
    - Flattens the output of the convolutional layers into a 1D vector for feeding into the fully-connected layers.
  + **Dense Layers:**
    - One fully-connected layer with 64 neurons and ReLU activation.
    - Dropout layer with a rate of 0.5 to prevent overfitting.
    - Final output layer with 7 neurons (corresponding to the number of currency classes) and softmax activation for probability distribution across classes.
* **Training Process:**
  + **Image Loading and Preprocessing:**
    - image.load\_img is used to load an image for prediction.
    - The image is resized to target\_size and converted to a NumPy array.
    - The array is normalized by dividing by 255 for color scaling between 0 and 1.
    - np.expand\_dims adds an extra dimension to convert the single image into a batch-like format expected by the model.
  + **Hyperparameters:**
    - Number of epochs (iterations): 30
    - Initial learning rate: 1e-3
    - Batch size: 32
  + **Optimizer:**
    - Adam optimizer is used for gradient descent with learning rate decay.
  + **Loss Function and Metrics:**
    - Categorical crossentropy is used as the loss function for multi-class classification.
    - Accuracy is used as the evaluation metric to measure the proportion of correct predictions.
  + **Model Fitting:**
    - model.fit trains the model on the training data generator.
    - steps\_per\_epoch is calculated to ensure all training data is used in each epoch.
    - Validation data generator is used to monitor performance on unseen data during training.
    - validation\_steps is calculated to iterate over the entire validation data.
  + **Training Plots:**
    - Training and validation accuracy/loss plots are generated using matplotlib to visualize the learning process.
* **Evaluation and Deployment:**
  + **Model Saving:**
    - The trained model weights are saved using model.save\_weights('Project\_weights.h5').
    - The entire model can potentially be saved using model.save('Project.h5')
  + **Model Loading:**
    - The model can be loaded back using model.load\_weights('Project\_weights.h5').
  + **Prediction:**
    - A function prepare is defined to preprocess an image for prediction.
    - The model predicts the probability of what note it could be.

**Additional Considerations and Improvements**

* **Data Augmentation:** The current data augmentation techniques are a good starting point. Consider exploring additional techniques like color jittering, random cropping, and lighting variations to further improve model robustness.
* **Class Imbalance:** If our dataset has imbalanced classes (unequal number of images per currency), it can affect training. Techniques like oversampling or undersampling the minority class or using weighted loss functions can be explored.
* **Transfer Learning:** Consider using pre-trained models like VGG16 or ResNet as feature extractors and fine-tuning them on our currency classification task. This can leverage existing knowledge and potentially improve performance, especially with limited data.
* **Hyperparameter Tuning:** The current hyperparameters (epochs, learning rate, batch size) can be further optimized using techniques like grid search or random search to find the best configuration for our specific dataset.
* **Evaluation Metrics:** While accuracy is a common metric, consider using additional metrics like precision, recall, and F1-score to get a more comprehensive understanding of the model's performance, especially for imbalanced classes.

**Further Exploration**

* **Real-time Detection:** Explore integrating the model with computer vision libraries like OpenCV to enable real-time currency detection in video streams.
* **Fake Currency Detection:** Extend the model to not only classify currency but also identify counterfeit notes by incorporating additional features or using techniques like anomaly detection.
* **Multi-Currency Support:** If more currencies are to be supported, retrain the model with an expanded dataset containing images of additional currencies.

**Conclusion**

This report provides a detailed breakdown of our currency detection model. We built a solid foundation with data augmentation, convolutional neural networks, and essential training practices. By considering the suggestions for improvement, deployment strategies, and further exploration, we can enhance our model's performance, scalability, and real-world applications.