International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

AI- Powered Visual Assistant for Text Reading and Indian Currency Recognition to Visually Impaired

Sneha Kumari[1], Ms. Sujitha S[2], Puja Sharma[3]

[1,2,3]Department of AIML
SJB Institute of Technology Bengaluru, India

Abstract
Visually impaired individuals often face challenges in reading printed text and identifying currency. This project introduces an Android-based AI vision assistant that provides audio and haptic feedback through two modes: Read Mode and Currency Mode. Read Mode uses Google ML Kit offline OCR to extract printed text and convert it into speech using text-to-speech technology. Currency Mode identifies six Indian denominations: ₹10, ₹20, ₹50, ₹100, ₹200 and ₹500.

The currency classification model uses MobileNetV2 transfer learning with 1,505 images, divided into training, validation and testing sets using stratified sampling. Data augmentation was applied to improve robustness against real-world image variations. Training was performed in two stages: initially, the MobileNetV2 base was frozen while the classification layers were trained. Subsequently, the final 30 layers were fine-tuned using a lower learning rate, with Batch Normalization layers kept frozen.

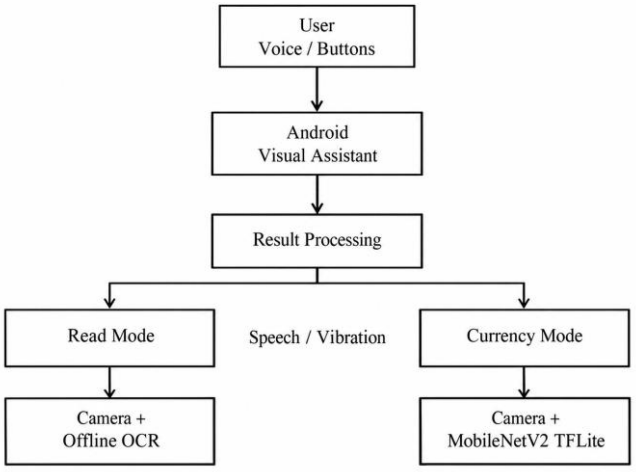
1. INTRODUCTION

Assistive technology could be used to provide a means of access to information for disabled individuals. Smartphones are a convenient way to provide assistive technology to visually impaired individuals. They have many useful components such as a camera, processor, microphone, speaker and vibratory motor. This project provides an Android visual assistance application that provides the user with non-visual information based on the camera. It has two main modes. The first mode allows the user to point their camera at English text, and the application will extract the text, recognise it and then convert it into speech. This is known as Read Mode. The second mode will take an image and determine what denomination of Indian currency it represents out of the 6 different currency types. The application has incorporated speech, vibration, and volume buttons to make it easy to navigate.

The main machine learning component of this project is the currency recogniser. The model uses MobileNetV2 as its CNN backbone due to its efficient design, meaning it can be used on mobile devices [1]. The model was trained using transfer learning to utilise the features learnt from the ImageNet dataset to classify the currencies from the 6 Indian denominations. After evaluating the model, the trained network was converted into a TensorFlow Lite model.

2. Related Work

Computer vision and machine learning-based assistive systems help visually impaired individuals access visual information. For example, Microsoft's Seeing AI uses accessible outputs to describe surroundings, making visual information easier to understand [2].



Optical Character Recognition (OCR) is essential for assisting visually impaired users in accessing printed text [3]. In Read Mode, OCR extracts text captured by the smartphone camera and converts it into audio feedback. For currency recognition, MobileNetV2 is used due to its efficient architecture, which combines inverted residual blocks, linear bottlenecks and depthwise separable convolutions to balance accuracy and computational efficiency [1]. Combined with TensorFlow Lite and smartphone camera input, it enables on-device currency recognition.

3. Proposed System

This work combines Artificial Intelligence, computer vision, and mobile computing to develop an assistive solution that supports visually impaired individuals in recognizing text and identifying currency. As demonstrated by Fig. 1, the application will run with ease by having the user choose a mode of operation, capture an image from user's smartphone camera and have the processed information returned either via audio, haptic feedback or both.

A. Read Mode

This module is designed to detect and extract English text from printed sources. The user's phone will take a picture of the text, and then the offline text recognition feature from Google ML Kit will process it. Using the text-to-speech engine, if the text has been recognised, the text will be read out. In cases where text recognition is unsuccessful, the system alerts the user through vibration and voice feedback, prompting them to recapture the image. The reading session is managed to ensure smooth interaction.

B. Currency Mode

The Currency Mode helps to detect various denominations of Indian currencies. The captured image is scaled to a size of 224 X 224. The image is then converted into three RGB channels to be fed into the TFLite model. The model generates probability scores for each of the six categories. The category with the highest probability is chosen as the output. The detected denomination is then provided to the user as audio and vibrations. This enables users to operate the application without relying on visual interaction with the screen.

4. TECHNOLOGY STACK

The system was developed using Python for machine learning model development and Kotlin for Android application development. The primary technologies and tools used in the implementation are summarized in Table I.

TABLE I.
PRINCIPAL TECHNOLOGIES AND CONFIGURATIONS

Technology	Role / Configuration
Python	ML development and experimentation; version 3.10.11
TensorFlow / Keras	Model training and conversion
MobileNetV2	ImageNet-pretrained transfer-learning backbone
TensorFlow Lite	On-device currency inference
Kotlin	Android application development
Jetpack Compose	Android user interface
CameraX	Camera access
Google ML Kit	Offline text recognition
OpenCV / EasyOCR	Prototype computer-vision/OCR experimentation
Gradle	Android build automation; version 9.5.0
Android SDK	Compile/target SDK 37; minimum SDK 26
ADB	Device installation and Logcat testing

TABLE II.
CURRENCY DATASET DISTRIBUTION

Subset	Images	Percentage
Training	1,053	70%
Validation	226	15%
Testing	226	15%
Total	1,505	100%

The currency dataset contained 1,505 images of six Indian denominations (₹10, ₹20, ₹50, ₹100, ₹200 and ₹500), divided into 1,053 training, 226 validation and 226 testing images using stratified sampling.

All images were resized to 224 × 224 pixels and converted to RGB format. MobileNetV2 preprocessing normalized pixel values to approximately −1 to +1. Data augmentation, including rotation (0.05), zoom (0.08), translation (0.08), contrast (0.15) and brightness (0.15), was applied to improve robustness against variations in smartphone image capture conditions.
The training pipeline is represented as:
I → Decode → Resize (224 × 224) → Float32 → MobileNetV2 Preprocessing (1)

TABLE III.
MOBILENETV2 CONFIGURATION

Parameter	Value
Architecture	MobileNetV2
Pretrained weights	ImageNet
Input	224 × 224 × 3
Include top	False
Output classes	6
Output activation	Softmax

5. MOBILE NET V2 MODEL AND TRAINING

A. Model Selection

As the aim is to deploy the model on an Android application, MobileNetV2 was selected as the backbone due to high efficiency in computation. MobileNetV2 is well-suited for mobile applications due to its lightweight architecture and transfer learning capabilities. It also supports efficient conversion to TensorFlow Lite for deployment on Android devices [1]. The model was initialized with pre-trained ImageNet weights, after which its original classification layer was replaced to support the required currency classification task. The model takes in 224 x 224 x 3 images. A new classification layer with 6 classes and a softmax activation was added.

B. Initial Transfer Learning

In the initial training phase, the pre-trained MobileNetV2 backbone will be frozen, therefore the image features will not change. The Adam optimiser is used to train the model with learning rate of (1×10^{-3}). The sparse categorical cross-entropy function is used to calculate the loss and the accuracy metric. Training was performed for up to 20 epochs to optimize the newly added classification layers for currency recognition.

C. Fine-Tuning

We then make the model trainable and fine-tune the last 30 layers using a lower learning rate (1×10^{-5}). This results in the first part of MobileNetV2 being frozen, along with the Batch Normalisation Layers. There are 19 trainable layers. We will only fine-tune the model for up to 15 epochs. During this phase, we achieved a training accuracy of 95.44%, with a loss of 0.1331. The validation accuracy achieved was 87.17%, with a loss of 0.4046. Early stopping was used to return to the best model weights.

D. Training Control

Three callbacks have been added. The first callback is EarlyStopping, which will stop training and revert to the weights of the best epoch based on the validation loss. The second is ModelCheckpoint, which will save the model if there has been improvement in validation loss. Lastly, ReduceLROnPlateau will reduce the learning rate if there is no improvement in validation loss. This helps with managing the convergence of the model, retaining the improved weights and preventing training beyond a point where no improvements have been made.

6. Evaluation

During testing on an Android device, a preprocessing issue was identified. The application normalized RGB pixel values before passing them to the classifier, causing duplicate preprocessing because MobileNetV2 preprocessing was already included in the model. This resulted in a strong bias towards predicting ₹10. The issue was resolved by modifying the Android input pipeline to pass RGB values in their original range (0–255), allowing the model to handle preprocessing internally. After this correction, all six currency denominations were successfully classified. This highlights the importance of consistent preprocessing during mobile model deployment, beyond offline accuracy.

A. Held-Out Test Performance

The final model achieved a loss of 0.3175 and an accuracy of 0.8761 on the held-out test set with 226 images. This means that it has a classification accuracy of 87.61%, meaning it is predicting the correct currency denomination of the test images. This provides an independent assessment of the model's performance on images that were not included in the training or validation datasets.

TABLE IV.
REPORTED MODEL PERFORMANCE

Evaluation	Result
Training accuracy after fine-tuning	95.44%
Validation accuracy	87.17%
Held-out test accuracy	87.61%
Held-out test loss	0.3175
Real-world correct predictions	121 / 150
Real-world accuracy	80.67%

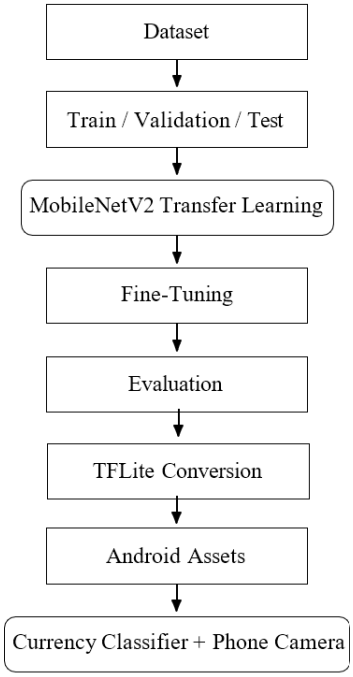


Fig. 2. Currency-recognition development and deployment pipeline.

B. Real-World Evaluation

A further dataset containing 150 real images (25 for each of the 6 currencies supported) was created to test the model. Of these 150 images, 121 were correctly identified by the model, meaning the accuracy achieved was 80.67%. This is a decrease compared to the held-out test set, where an accuracy of 87.61% was achieved. This could be due to the model being more susceptible to variation in the images when taken in real life, such as different lighting, backgrounds, note orientation, distance from the camera and the visual quality of the currency note.

C. TensorFlow Lite Conversion and Verification

Once trained and evaluated, the Keras model is converted to TensorFlow Lite. This has been saved to a file named currency_v2_mobilenetv2.tflite. The file is around 8.5 MB in size. It takes in Float32 tensors of size 224×224×3. It then produces 6 Float32 outputs, one for each class of currency that is supported. Before using this model inside the Android app, it is run on its own on a computer to make sure it produces the right outputs. Running the Keras model and TFLite model separately can help with identifying any differences between the two models.

D. Android Application Implementation

The Android application was developed using Kotlin, Jetpack Compose, Android camera, speech, and vibration APIs and the LiteRT/TensorFlow Lite runtime. The code is broken down into several components to manage app states, interactions, camera features, and text recognition. These components include: MainActivity, App Interaction Manager, Mode Manager, Interaction Controller, Volume Button Handler, Camera Screen, Offline Text Recognition Manager, Currency Classifier, Speech Manager, and

Vibration Manager. These work together to provide a way for users to control which mode they'd like, take photographs, perform offline text recognition, classify currencies, and provide verbal responses. The application is organized into smaller, independent components, separating the user interface from the text recognition functionality to improve maintainability.

7. CONCLUSION

This study presents an AI-based Android visual assistance application designed to help visually impaired individuals access visual information through audio and haptic feedback. The application includes two modes: Read Mode, which uses offline Google ML Kit OCR and text-to-speech to read printed text, and Currency Mode, which identifies six Indian currency denominations using a MobileNetV2-based classification model. Trained on 1,505 images using transfer learning, data augmentation and fine-tuning, the model achieved 87.61% accuracy on the test dataset and 80.67% on 150 real-world images. The model has a size of approximately 8.5 MB and uses a 224 × 224 × 3 Float32 input, with TensorFlow Lite enabling on-device deployment. Overall, the application provides a basic prototype that can be further improved through better recognition accuracy, enhanced accessibility and support for detecting multiple currency notes in a single image.

ACKNOWLEDGMENT

We would like to thank the Department of Artificial Intelligence and Machine Learning, SJB Institute of Technology, Bengaluru, for providing us with an academic environment to work on this project.

REFERENCES

1. M. Sandler et al., "MobileNetV2: Inverted Residuals and Linear Bottlenecks," Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 4510–4520, doi: 10.1109/CVPR.2018.00474.
2. Microsoft Accessibility, "Seeing AI: New Technology Research to Support the Blind and Visually Impaired Community."
3. Q. Gao, R. Manduchi, P. Y. Ramulu, G. E. Legge, et al., "VI-OCR: 'Visually Impaired' OCR-Based Text Recognition Process for Accessibility," Scientific Reports, vol. 16, p. 1269, 2026.
4. TensorFlow/Keras documentation, "MobileNetV2, preprocessing, model training and TensorFlow Lite conversion."
5. Android Developers, "Android application development, Jetpack Compose, CameraX, permissions, and Gradle builds."
6. Google, "ML Kit Text Recognition documentation."
7. Android Developers, "Android SDK Platform Tools documentation for ADB and Logcat."