CHAPTER 1
ABSTRACT

Art Guess is an Android application that combines drawing and machine learning to create an engaging and educational game. Art Guess challenges users to draw a given word within a time limit. The application employs a pre-trained neural network model to analyze the user's drawing and determine whether it accurately represents the provided word.

The game's user interface presents a canvas where users can freely draw using their finger or a stylus. Upon selecting the "Start" button, a random word is displayed, and the countdown begins. As the user draws, the neural network continuously evaluates the strokes and provides real-time feedback on the accuracy of the drawing. Once the time limit expires, the neural network makes a final prediction, and the result is conveyed to the user through both a text and speech synthesis.

Art Guess aims to make learning and practicing drawing skills more engaging and interactive. By combining machine learning techniques with a user-friendly interface, the application encourages users to improve their drawing abilities while providing instant feedback and reinforcement. The integration of speech recognition further enhances the user experience, making the game accessible to a wider audience.

CHAPTER 2
INTRODUCTION

"Art Guess" is an innovative Android application that combines the creative process of drawing with the power of neural networks for word prediction. Users are presented with a canvas and a specific word prompt, along with a time limit. Their task is to draw the given word to the best of their ability within the allotted time. Once the drawing is submitted, the application utilizes a convolutional neural network (CNN), implemented using TensorFlow, to predict the word based on the user's drawing. The application's CNN model continuously processes the canvas image and attempts to predict the word being drawn.

The CNN model employed in Art Guess is trained on a dataset consisting of 123 categories, enabling it to recognize a diverse range of objects and concepts. The model's predictions are conveyed to the user through both text and speech feedback, providing real-time updates on the model's interpretation of the user's drawing. The application keeps track of the user's performance, providing a score or rating based on the accuracy of the CNN model's predictions.

Art Guess leverages the capabilities of TensorFlow, a widely-used open-source machine learning framework, for implementing and training the CNN model. The application seamlessly integrates the trained model into the Android environment, enabling real-time image recognition and prediction on the user's drawings.

This project not only offers an engaging and interactive drawing experience but also serves as an educational tool for users to explore the capabilities of neural networks in drawing recognition tasks.

By leveraging the power of deep learning, Art Guess enhances the user's understanding of neural networks while providing entertainment and creative expression. It merges artistry with technology in a fun and educational manner, making it an exciting addition to the realm of mobile applications.

CHAPTER 3
REQUIREMENT SPECIFICATION

3.1 HARDWARE SPECIFICATION

Processor	: Intel(R) Core(TM) i5
Memory(RAM)	: 8.00 GB

3.2 SOFTWARE SPECIFICATION

Operating system	: Windows 10
Language use	: Java,Python
Front end	: Xml
IDE	: Android Studio, Google colab
Framework	: Tensorflow
Software type	: Android Application
Jdk version	: 21
Gradle version	: 6.7.1

Android Device or Emulator: Samsung SM-A515F Android13.0(“tiramisu")

3.3 FUNCTIONAL REQUIREMENTS

User Interface:

Allow users to select the desired number of doodles to draw

Canvas: Provide a canvas interface for users to draw doodles.

Word Prompt: Display a word prompt for users to draw within a specified timelimit.

Timer: Implement a timer to indicate the remaining time for drawing.

Drawing Interaction:

Drawing Tools: Offer basic drawing tools such as a pencil, eraser, and color selection.

Drawing Recognition: Recognize and interpret user doodles in real-time.

Prediction and feedback:

CNN Integration: Integrate a Convolutional Neural Network (CNN) model for predicting user drawings.

Prediction Accuracy: Ensure high accuracy in predicting user drawings based on provided words.

Feedback Mechanisms: Provide textual and speech feedback to users regarding their drawing accuracy.

User Interaction:

User Input: Allow users to interact with the application through touch input on the canvas and UI elements.

Feedback Display: Present feedback in a clear and understandable manner to the user.

User Controls: Offer controls for users to start, pause, and reset drawing sessions.

Performance:

Optimization: Optimize the application for smooth performance on Android devices of varying specifications.

Real-time Prediction: Ensure real-time prediction of user drawings to maintain user engagement.

Data management:

Dataset Handling: Manage the dataset effectively for training and testing purposes.

Accessibility:

Accessibility Features: Include features to accommodate users with disabilities, such as voice input and screen reader support.

UI Clarity: Ensure clarity and readability of UI elements for users with visual

Security:

Data Privacy: Maintain user privacy and confidentiality of drawing data within the application.

Secure Communication: Ensure secure communication between the application and any external servers or APIs.

3.4 NON-FUNCTIONAL REQUIREMENTS:

Scalability:

The TensorFlow Lite model should be scalable to accommodate future updates or expansions of the doodle categories.

Reliability:

The application should be robust and stable, with minimal crashes or unexpected behavior.

The TensorFlow Lite model should produce accurate predictions consistently, without significant variability in performance.

Usability:

The user interface should be intuitive and easy to navigate, with clear instructions and prompts for drawing.

The application should support accessibility features, such as voice input or screen reader support, to accommodate users with disabilities.

Compatibility:

The application should be compatible with a wide range of Android devices, operating system versions, and screen sizes.

The TensorFlow Lite model should be compatible with different hardware configurations, including devices with varying levels of computational resources.

Resource Usage:

The application should use resources efficiently, including CPU, memory, and battery, to minimize drain on device resources.

The TensorFlow Lite model should have a small memory footprint and low computational overhead to maximize performance on mobile devices.

Maintainability:

The codebase should be well-structured and documented, facilitating future maintenance and updates by developers.

The TensorFlow Lite model should be modular and easily upgradable, allowing for the integration of new features or improvements.

CHAPTER 4
DATASETS

The dataset used for training the Convolutional Neural Network (CNN) model in the Art Guess application is derived from the Quick, Draw! project. The Quick, Draw! dataset is a collection of millions of drawings contributed by people around the world through a game-like interface developed by Google.
The Quick, Draw! Dataset has a collection of 50 Million drawings across 345 categories. This data is made available by Google, Inc. under the Creative Commons Attribution 4.0 International license. Google Creative Cloud: Quick, Draw! Dataset.

The Quick, Draw! dataset provides the sketches in a 28x28 grayscale bitmap in NumPy format (.npy), which is a format of a highly popular Python library, namely, NumPy , that provides both multi-dimensional arrays and matrices, and a large collection of high-level mathematical functions to operate on these data structures. Therefore, the dimension of the input was 28*28*1. Drawings for each class in the constructed dataset were extracted from these provided NumPy files and stored as image files.

For the Art Guess application, a subset of the Quick, Draw! dataset consisting of 123 categories of images was selected. Each category represents a different object or concept that users may be prompted to draw within the application. The dataset is stored in the .npy format, which is a binary format used for efficiently storing and loading large arrays of numerical data in Python.

Each image in the dataset is represented as a 28x28 greyscale pixel array, where each pixel value represents the intensity of the gray scale colour. The images are pre-processed and normalized before being fed into the CNN model for training. Additionally, the dataset is split into training, validation, and testing sets to evaluate the performance of the model accurately.

Dataset Details:

Total Categories: 123

Images per Category: 6000 (4920 for training, 1080 for testing)

Total Training Images: 606,360 (4920 images * 123 categories)

Total Testing Images: 132,840 (1080 images * 123 categories)

Training and Testing:

Training Data: The CNN model is trained on 4920 images from each of the 123 categories, resulting in a total of 606,360 training images. These images are used to optimize the model's parameters and learn the relationships between different doodle categories.

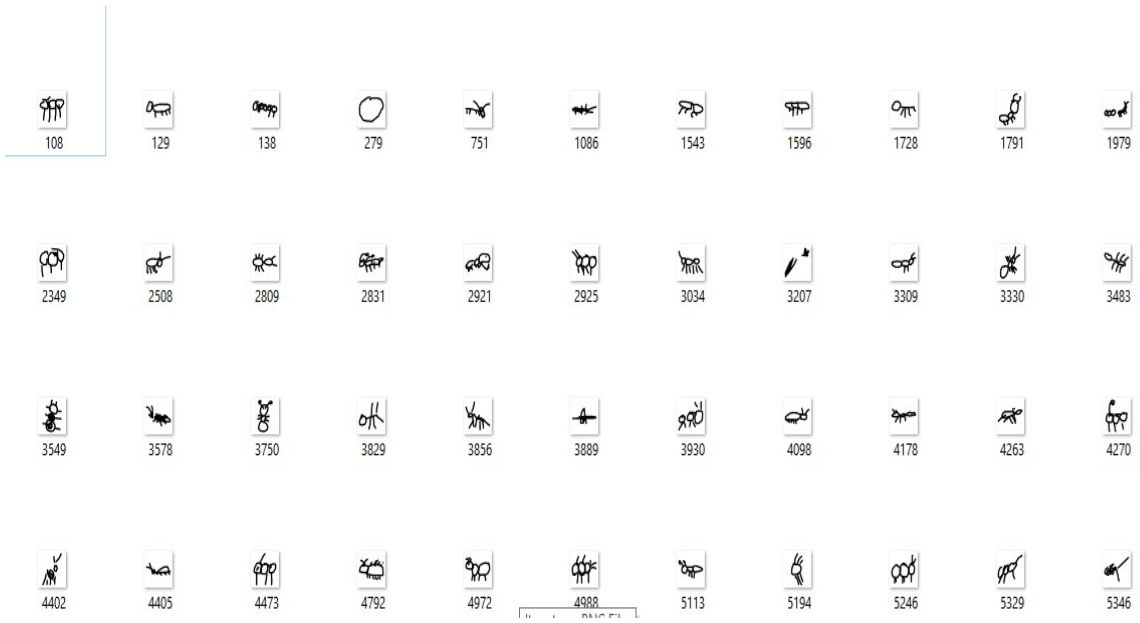
Testing Data: To evaluate the performance of the trained model, testing is conducted on 1080 images from each of the 123 categories, totaling 132,840 testing images. The model's predictions on these unseen test images are compared against the ground truth labels to assess its accuracy and generalization capability.

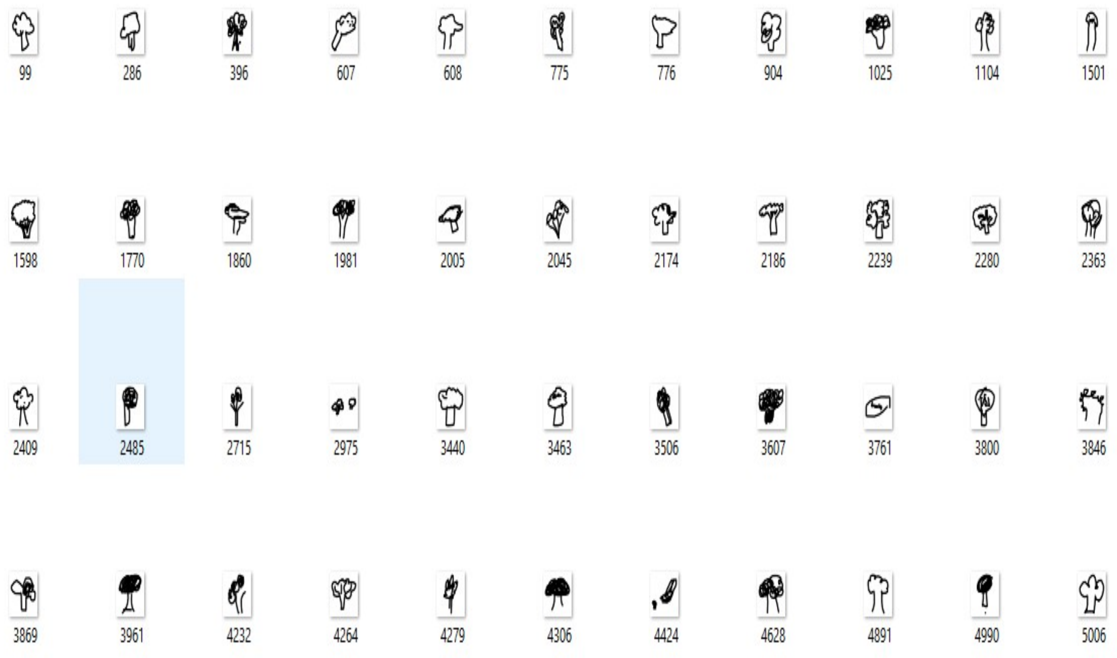
Dataset categories

The Eiffel	Brain	Dragon
Tower	Broccoli	Duck
The Mona Lisa	Bulldozer	Elephant
Airplane	Butterfly	Envelope
Ambulance	Cactus	Eye
Angel	Calculator	Face
Ant	Camel	Feather
Apple	Castle	Fish
Axe	Ceiling fan	Flip flops
Banana	Circle	Flower
Basket	Compass	Frog
Beach	Cow	Giraffe
Bicycle	Cruise ship	Grapes
Birthday cake	Cruise ship	Grass
Book	Cup	Guitar
Boomerang	Dolphin	Hammer

Hand
Hat
Helicopter
Hospital
House
Ice cream
Jacket
Jail
Kangaroo
Key
Keyboard
Ladder
Leaf
Leg
Light bulb
Lightning
Lion
Lipstick
Lollipop
Map
Mermaid
Mosquito
Mountain
Mouse
Mushroom
Necklace
Nose
Ocean
Octopus
Onion
Pants
Parrot
Pencil
Piano
Pig
Pillow
Pizza
Pond
Rabbit
Radio
Rain
Rainbow
Rifle
Rollerskates
Sandwich
Saw
Scissors
Sea turtle
See saw
Shark
Skull
Spider
Stairs
Star
Stereo
Strawberry
Swan
Sword
T-shirt
Teddy bear
Television
Tent
Tiger
Toothbrush
Tree
Umbrella
Underwear
Vase
Washing machine
Whale
Wheel
Windmill
Wine glass
Wristwatch
Zebra
Zigzag

Dataset images



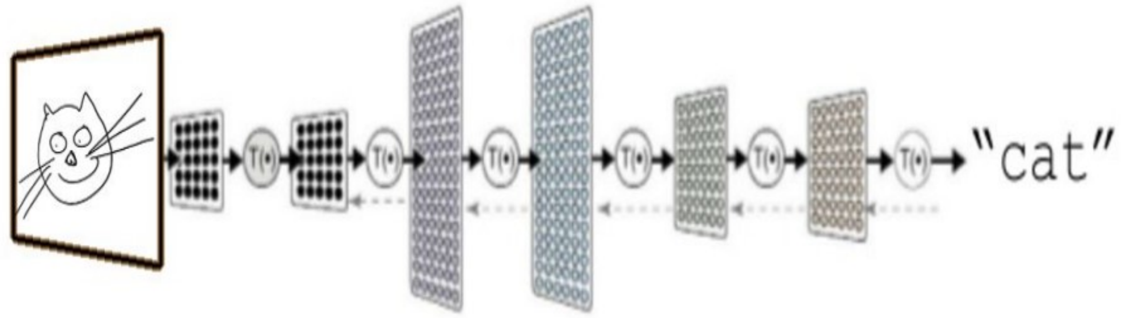


CHAPTER 5
METHODS AND ALGORITHMS USED

Convolutional Neural Network

Deep Learning has proved to be a very powerful tool because of its ability to handle large amounts of data. The interest to use hidden layers has surpassed traditional techniques, especially in pattern recognition. One of the most popular deep neural networks is Convolutional Neural Networks (also known as CNN or ConvNet) in deep learning, especially when it comes to Computer Vision applications.

In deep learning, a **convolutional neural network** (**CNN/ConvNet**) is a class of deep neural networks, most commonly applied to analyze visual imagery. Now when we think of a neural network we think about matrix multiplications but that is not the case with ConvNet. It uses a special technique called Convolution. Now in mathematics convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.



The Art guess is based on a convolutional neural network and responsible for both learning the stroke patterns of sketches, and predicting the classes of the given sketches. For the implementation of the proposed model, a widely-used, open-source high-level deep neural networks library written in Python, namely, Keras , was utilized. Keras is capable of running on the top of various backends such as TensorFlow . Within this study, TensorFlow, which is an open-source machine learning platform developed by Google, was opted as the backend because of being the recommended one by the developer of

Keras . The proposed model was trained on the Google Colab platform since this platform provides powerful GPUs, such as the Nvidia Tesla K80, which is a key necessity for the training of deep neural networks because of demanding too much computational power. The other advantages of using the Google Colab are: The seamless integration with the Google Drive, which provides permanent storage for the datasets, and the platform already provides various highly popular Python libraries related to data science including but not limited to TensorFlow, Keras, scikit-learn, and matplotlib. In addition to this, additional packages can be installed through the pip, which is the package installer for Python.

The proposed model starts with a convolution layer (denoted with Conv2D) which accepts the input and performs the convolution operations on. After the first sequential convolutional layers, a Maxpooling layer, which is responsible for progressively reducing the spatial size of the representation that helps to reduce the number of parameters and computation in the network , was employed. Dropout is a widely-used technique that was proposed to prevent the 'overfitting' problem, which is one of the biggest challenges of deep neural networks . In order to prevent overfitting, several Drop out layers were employed in various positions. In addition to this, Max ppoling layers also help to control overfitting . Batch Normalization (denoted with norm) layers were employed to normalize the activations of the previous layer in each batch. After the convolutions were performed, the dimension transformation was carried out by the Flatten layer, which prepares the data to be passed to the dense layers. Then, the two Dense layers, which are deeply connected neural network components, were employed. The last layer, namely, the Softmax layer, was responsible for the classification of the given sketch as one of the pre-defined nine classes. The Rectified Linear Units (ReLU) was employed as the activation function for all the activations except for the last layer since it provided the best accuracy during the hyper-parameter optimization.

THE DETAILED INFORMATION ABOUT EACH LAYER IN THE PROPOSED MODEL

No	Layer name	Layer parameters
1	Conv2D	input_shape=(28,28,1), filters=16, kernel_size=(3,3), activation='relu', kernel_initializer='uniform'
2	Max pooling	pool_size=(2,2), strides=(2,2)
3	Dropout	
4	Conv2D	filters=16, kernel_size=(3,3), activation='relu', kernel_initializer='uniform')
5	Max pooling	pool_size=(2,2), strides=(2,2))
6	Dropout	rate=0.5
7	Flatten	
8	Dense	units=256, activation='relu', kernel_initializer='uniform'
9	Dense	units=123, activation='softmax', kernel_initializer='uniform'

Tensorflow:

TensorFlow, an open-source machine learning framework developed by Google, played a crucial role in the Art Guess project. TensorFlow provides a comprehensive platform for building and training machine learning models, particularly deep neural networks like Convolutional Neural Networks (CNNs), which are essential for image recognition tasks. In this project, TensorFlow was utilized to implement and train the CNN model responsible for predicting user doodle drawings based on provided words. TensorFlow's extensive set of APIs and tools enabled efficient development and optimization of the CNN architecture, allowing for seamless integration with the Android application. Through TensorFlow's high-level APIs, such as Keras, the CNN model was constructed layer by layer, specifying convolutional layers, pooling layers, and fully connected layers to capture and learn relevant features from the doodle images. Additionally, TensorFlow's powerful GPU acceleration capabilities facilitated faster training times, enabling the model to learn from the extensive Quick, Draw! dataset efficiently. Furthermore, TensorFlow's compatibility with Android development enabled seamless deployment of the trained model within the Art Guess application, ensuring real-time prediction and feedback during user drawing sessions. Overall, TensorFlow served as the backbone of the machine learning pipeline in the Art Guess project, empowering the application to provide users with an immersive and accurate drawing experience while leveraging state-of-the-art deep learning techniques.

TensorFlow Lite:

It an extension of TensorFlow, is a lightweight solution specifically designed for deploying machine learning models on mobile and embedded devices. In the Art Guess project, TensorFlow Lite played a pivotal role in integrating the trained Convolutional Neural Network (CNN) model into the Android application for real-time prediction of user doodle drawings. TensorFlow Lite offers several advantages for mobile deployment, including smaller model sizes, faster inference times, and compatibility with various hardware architectures commonly found in smartphones and tablets. By converting the trained TensorFlow model into a TensorFlow Lite format, the model could be optimized for mobile inference, ensuring minimal impact on device resources such as memory and battery life. Furthermore, TensorFlow Lite's support for hardware acceleration, including GPU and Neural Processing Unit (NPU) acceleration on compatible devices, allowed for efficient execution of inference tasks, enabling smooth and responsive user interactions within the Art Guess application. Through TensorFlow Lite, the CNN model could seamlessly integrate with the Android application, providing users with accurate and instantaneous feedback on their doodle drawings while maintaining optimal performance on mobile devices. Overall, TensorFlow Lite's versatility and efficiency made it an ideal choice for deploying machine learning models in mobile applications like Art Guess, enabling a seamless and immersive user experience.

Keras:

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, Theano, or Microsoft Cognitive Toolkit (CNTK). It was developed with a focus on enabling fast experimentation. Keras allows for easy and fast prototyping, supporting both convolutional networks and recurrent networks, as well as a combination of the two. It is particularly user-friendly, providing simple and intuitive interfaces to build and train deep learning models.

In the context of the Art Guess project, Keras was used to design, build, and train the Convolutional Neural Network (CNN) model for predicting user doodle drawings. Leveraging Keras's high-level abstraction, developers could define the CNN architecture with minimal code, specifying layers such as convolutional layers, pooling layers, and fully connected layers. Additionally, Keras offers a wide range of pre-built layers and modules, simplifying the process of constructing complex neural network architectures.

Keras also provides utilities for training models, including built-in support for various optimization algorithms, loss functions, and evaluation metrics. Developers could easily configure and customize these components to fine-tune the CNN model's performance and accuracy. Furthermore, Keras seamlessly integrates with TensorFlow, allowing for efficient execution of training and inference tasks on both CPU and GPU hardware.

CHAPTER 6
IMPLEMENTATION

Data Preprocessing:

Dataset Selection: The Quick, Draw! dataset was selected as the primary source of doodle images for training the Convolutional Neural Network (CNN) model.

Data Preparation: The dataset was preprocessed to extract a subset of 123 categories of doodle images, with each category containing 6000 images. Images were resized to a standardized dimension and converted to grayscale for consistency.

Dataset Splitting: The dataset was split into training and testing subsets, with 4920 images from each category used for training and 1080 images for testing.

DATASET LINK: https://github.com/jp-test-account/dataset.git

The Quick, Draw! dataset provides the sketches in a 28x28 grayscale bitmap in NumPy format (.npy), which is a format of a highly popular Python library, namely, NumPy , that provides both multi-dimensional arrays and matrices, and a large collection of high-level mathematical functions to operate on these data structures. Therefore, the dimension of the input was 28*28*1. Drawings for each class in the constructed dataset were extracted from these provided NumPy files and stored as image files.

Dataset Details:

Total Categories: 123

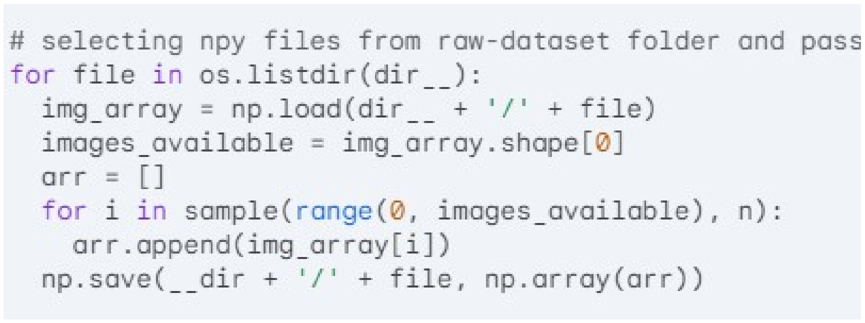
Images per Category: 6000 (4920 for training, 1080 for testing)

Total Training Images: 606,360 (4920 images * 123 categories)

Total Testing Images: 132,840 (1080 images * 123 categories)

npy to smaller npy

'npy-to-smaller-npy.py' converts randomly selected n (28,28) numpy array to and saves them todestined folder with n arrays



```
# selecting npy files from raw-dataset folder and pass
for file in os.listdir(dir__):
  img_array = np.load(dir__ + '/' + file)
  images_available = img_array.shape[0]
  arr = []
  for i in sample(range(0, images_available), n):
    arr.append(img_array[i])
  np.save(__dir + '/' + file, np.array(arr))
```

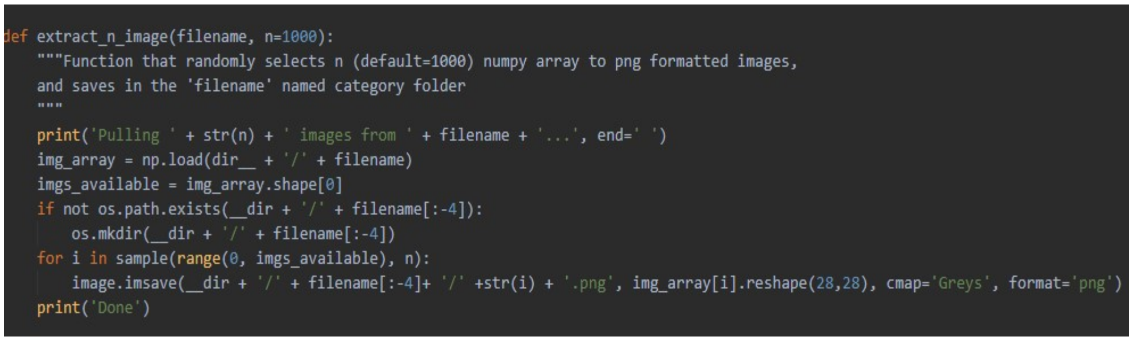
This loop iterates over the files in the source directory (dir__).

For each file:

- It loads the entire NumPy array using np.load.
- It gets the total number of images in the array using img_array.shape[0].
- It creates an empty list arr to store the randomly selected images.
- It uses sample(range(0, images_available), n) to randomly select n indices from the range without replacement (ensuring unique images).
- It iterates through the random indices and appends the corresponding images from img_array to arr.
- Finally, it saves the arr (containing the n randomly selected images) as a new NumPy array using np.save with the same filename in the destination directory (__dir).

npy to png:

'npy-to-png.py' converts randomly selected 1000 (28,28) numpy array to png image file and categorises it according to the npy file name



```
def extract_n_image(filename, n=1000):
    """Function that randomly selects n (default=1000) numpy array to png formatted images,
    and saves in the 'filename' named category folder
    """
    print('Pulling ' + str(n) + ' images from ' + filename + '...', end=' ')
    img_array = np.load(dir__ + '/' + filename)
    imgs_available = img_array.shape[0]
    if not os.path.exists(__dir + '/' + filename[:-4]):
        os.mkdir(__dir + '/' + filename[:-4])
    for i in sample(range(0, imgs_available), n):
        image.imsave(__dir + '/' + filename[:-4]+ '/' +str(i) + '.png', img_array[i].reshape(28,28), cmap='Greys', format='png')
    print('Done')
```

Model Development:

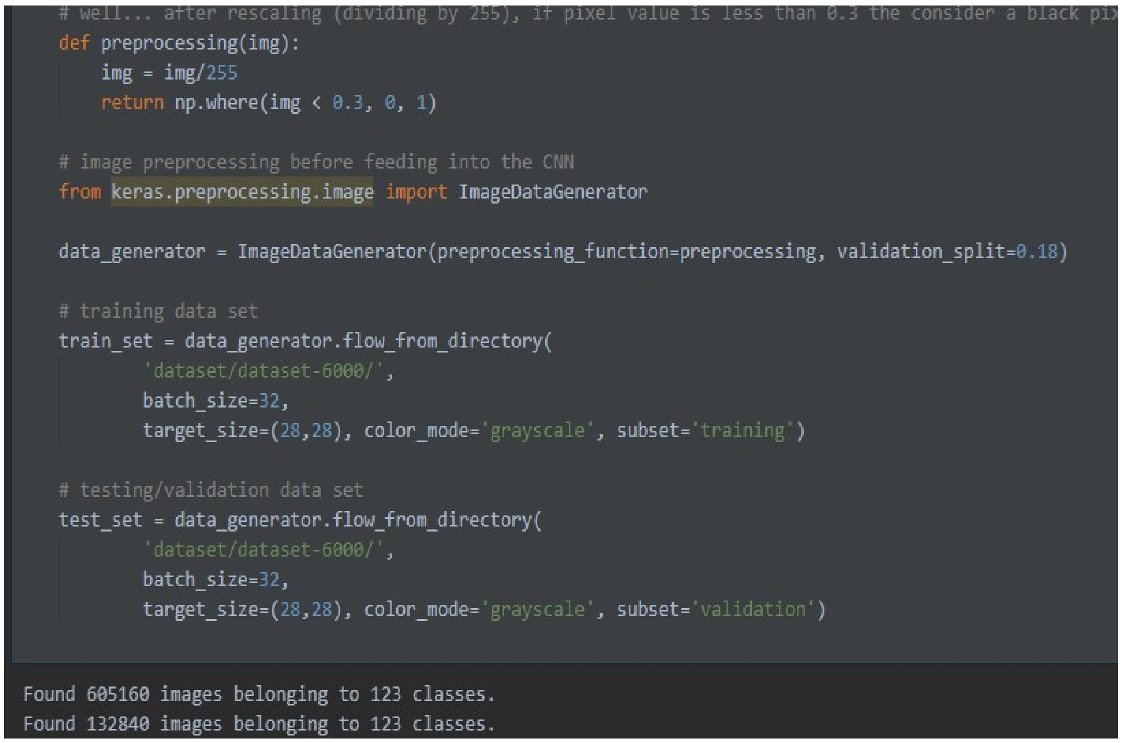
CNN Architecture: The CNN model was developed using the Keras API, built on top of TensorFlow. The architecture consisted of convolutional layers, max-pooling layers, and fully connected layers.

Training Procedure: The model was trained using the training dataset for an initial 5 epochs to establish baseline performance. Subsequently, the number of epochs was extended to 10 to further refine the model and improve prediction accuracy.

Optimization and Evaluation: Various optimization techniques, including batch normalization and dropout regularization, were applied to improve the model's generalization capability. The model's performance was evaluated using metrics such as accuracy, loss, and validation results.

The proposed model starts with a convolution layer (denoted with Conv2D) which accepts the input and performs the convolution operations on. After the first sequential convolutional layers, a Maxpooling layer, which is responsible for progressively reducing the spatial size of the representation that helps to reduce the number of parameters and computation in the network , was employed. Dropout is a widely-used technique that was proposed to prevent the 'overfitting' problem, which is one of the biggest challenges of deep neural networks . In order to prevent overfitting, several Drop out layers were employed in various positions. In addition to this, Max ppoling layers also help to control overfitting . Batch Normalization (denoted with norm) layers were employed to normalize the activations of the previous layer in each batch. After the convolutions were performed, the dimension transformation was carried out by the Flatten layer, which prepares the data to be passed to the dense layers. Then, the two Dense layers, which are deeply connected neural network components, were employed. The last layer, namely, the Softmax layer, was responsible for the classification of the given sketch as one of the pre-defined nine classes. The Rectified Linear Units (ReLU) was employed as the activation function for all the activations except for the last layer since it provided the best accuracy during the hyper-parameter optimization.

Image preprocessing before feeding into the neural net - this is done to keep only the important dark colored pixel and removing other noises around it



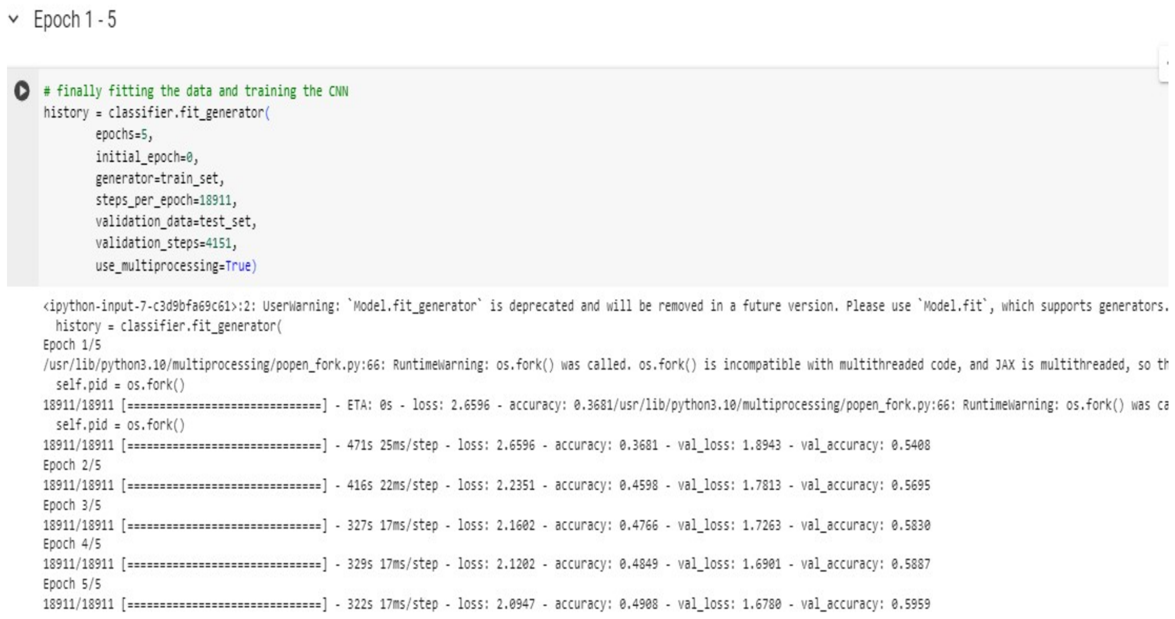
```
# well... after rescaling (dividing by 255), if pixel value is less than 0.3 the consider a black pix
def preprocessing(img):
    img = img/255
    return np.where(img < 0.3, 0, 1)

# image preprocessing before feeding into the CNN
from keras.preprocessing.image import ImageDataGenerator

data_generator = ImageDataGenerator(preprocessing_function=preprocessing, validation_split=0.18)

# training data set
train_set = data_generator.flow_from_directory(
        'dataset/dataset-6000/',
        batch_size=32,
        target_size=(28,28), color_mode='grayscale', subset='training')

# testing/validation data set
test_set = data_generator.flow_from_directory(
        'dataset/dataset-6000/',
        batch_size=32,
        target_size=(28,28), color_mode='grayscale', subset='validation')
```

```
Found 605160 images belonging to 123 classes.
Found 132840 images belonging to 123 classes.
```

Epochs:

Each time a dataset passes through an algorithm, it is said to have completed an epoch. Therefore, Epoch, in machine learning, refers to the one entire passing of training data through the algorithm. It's a hyperparameter that determines the process of training the machine learning model. The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters.

The training procedure for the Art Guess application involved training the Convolutional Neural Network (CNN) model on the selected dataset of doodle images. Initially, the model was trained for 5 epochs to establish baseline performance and optimize the model's parameters. Subsequently, based on the initial training results and performance evaluation, the number of epochs was extended to 10 to further refine the model and improve prediction accuracy.

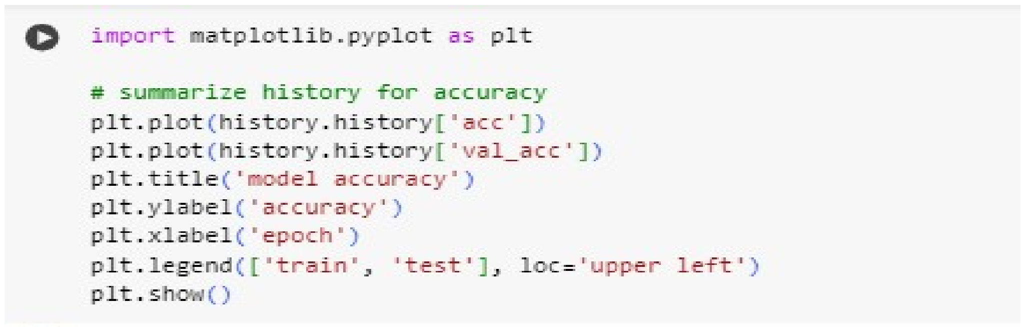


Epoch 1 - 5

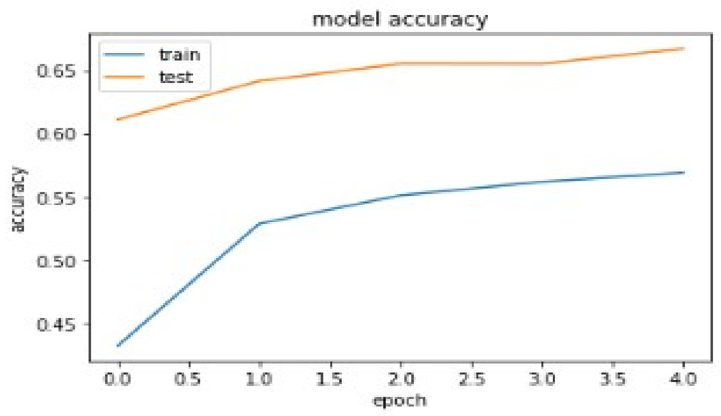
```
# finally fitting the data and training the CNN
history = classifier.fit_generator(
        epochs=5,
        initial_epoch=0,
        generator=train_set,
        steps_per_epoch=18911,
        validation_data=test_set,
        validation_steps=4151,
        use_multiprocessing=True)
```

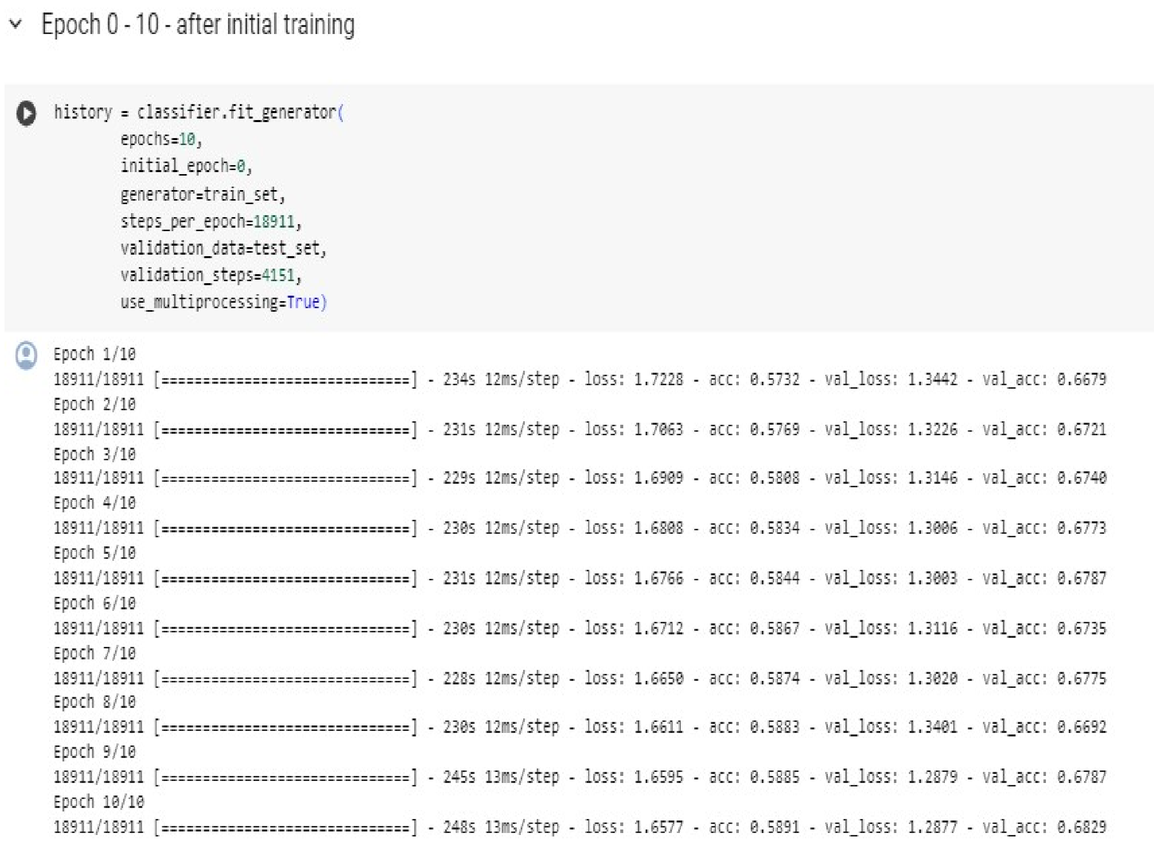
```
<ipython-input-7-c3d9bfa69c61>:2: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  history = classifier.fit_generator(
Epoch 1/5
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was called. os.fork() is incompatible with multithreaded code, and JAX is multithreaded, so th
  self.pid = os.fork()
18911/18911 [==============================] - ETA: 0s - loss: 2.6596 - accuracy: 0.3681/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was ca
  self.pid = os.fork()
18911/18911 [==============================] - 471s 25ms/step - loss: 2.6596 - accuracy: 0.3681 - val_loss: 1.8943 - val_accuracy: 0.5408
Epoch 2/5
18911/18911 [==============================] - 416s 22ms/step - loss: 2.2351 - accuracy: 0.4598 - val_loss: 1.7813 - val_accuracy: 0.5695
Epoch 3/5
18911/18911 [==============================] - 327s 17ms/step - loss: 2.1602 - accuracy: 0.4766 - val_loss: 1.7263 - val_accuracy: 0.5830
Epoch 4/5
18911/18911 [==============================] - 329s 17ms/step - loss: 2.1202 - accuracy: 0.4849 - val_loss: 1.6901 - val_accuracy: 0.5887
Epoch 5/5
18911/18911 [==============================] - 322s 17ms/step - loss: 2.0947 - accuracy: 0.4908 - val_loss: 1.6780 - val_accuracy: 0.5959
```

Model accuracy



```
import matplotlib.pyplot as plt

# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



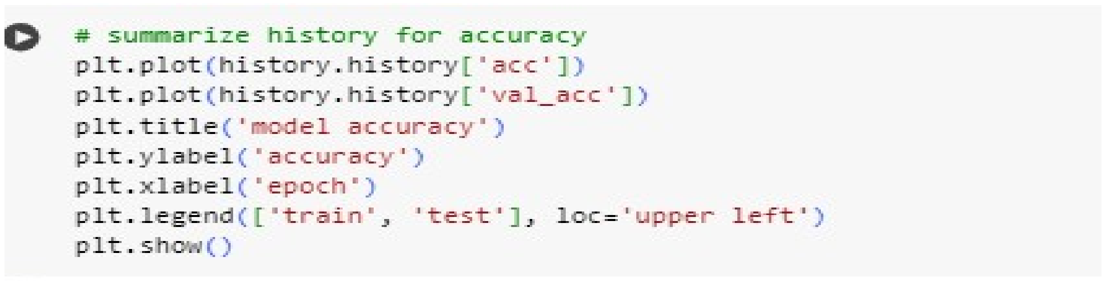


Epoch 0 - 10 - after initial training

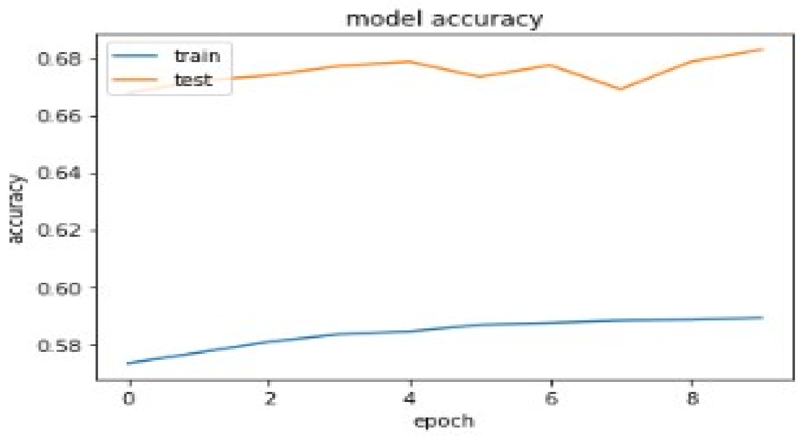
```
history = classifier.fit_generator(
        epochs=10,
        initial_epoch=0,
        generator=train_set,
        steps_per_epoch=18911,
        validation_data=test_set,
        validation_steps=4151,
        use_multiprocessing=True)
```

```
Epoch 1/10
18911/18911 [==============================] - 234s 12ms/step - loss: 1.7228 - acc: 0.5732 - val_loss: 1.3442 - val_acc: 0.6679
Epoch 2/10
18911/18911 [==============================] - 231s 12ms/step - loss: 1.7063 - acc: 0.5769 - val_loss: 1.3226 - val_acc: 0.6721
Epoch 3/10
18911/18911 [==============================] - 229s 12ms/step - loss: 1.6909 - acc: 0.5808 - val_loss: 1.3146 - val_acc: 0.6740
Epoch 4/10
18911/18911 [==============================] - 230s 12ms/step - loss: 1.6808 - acc: 0.5834 - val_loss: 1.3006 - val_acc: 0.6773
Epoch 5/10
18911/18911 [==============================] - 231s 12ms/step - loss: 1.6766 - acc: 0.5844 - val_loss: 1.3003 - val_acc: 0.6787
Epoch 6/10
18911/18911 [==============================] - 230s 12ms/step - loss: 1.6712 - acc: 0.5867 - val_loss: 1.3116 - val_acc: 0.6735
Epoch 7/10
18911/18911 [==============================] - 228s 12ms/step - loss: 1.6650 - acc: 0.5874 - val_loss: 1.3020 - val_acc: 0.6775
Epoch 8/10
18911/18911 [==============================] - 230s 12ms/step - loss: 1.6611 - acc: 0.5883 - val_loss: 1.3401 - val_acc: 0.6692
Epoch 9/10
18911/18911 [==============================] - 245s 13ms/step - loss: 1.6595 - acc: 0.5885 - val_loss: 1.2879 - val_acc: 0.6787
Epoch 10/10
18911/18911 [==============================] - 248s 13ms/step - loss: 1.6577 - acc: 0.5891 - val_loss: 1.2877 - val_acc: 0.6829
```

Model accuracy

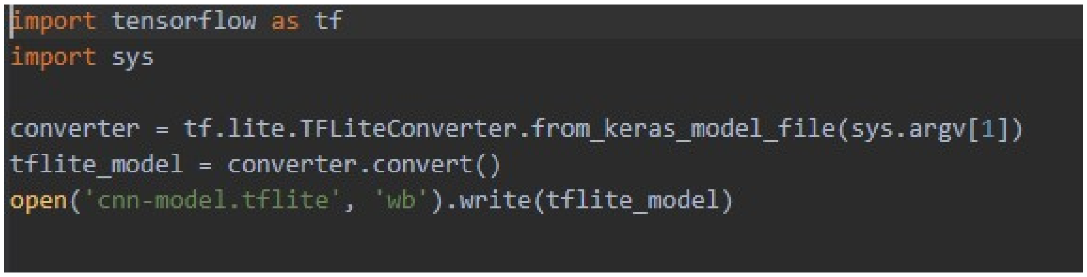


```
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Integration

TensorFlow Lite Conversion: Once the CNN model was trained and optimized, it was converted to TensorFlow Lite format for deployment on mobile devices.



```
import tensorflow as tf
import sys

converter = tf.lite.TFLiteConverter.from_keras_model_file(sys.argv[1])
tflite_model = converter.convert()
open('cnn-model.tflite', 'wb').write(tflite_model)
```

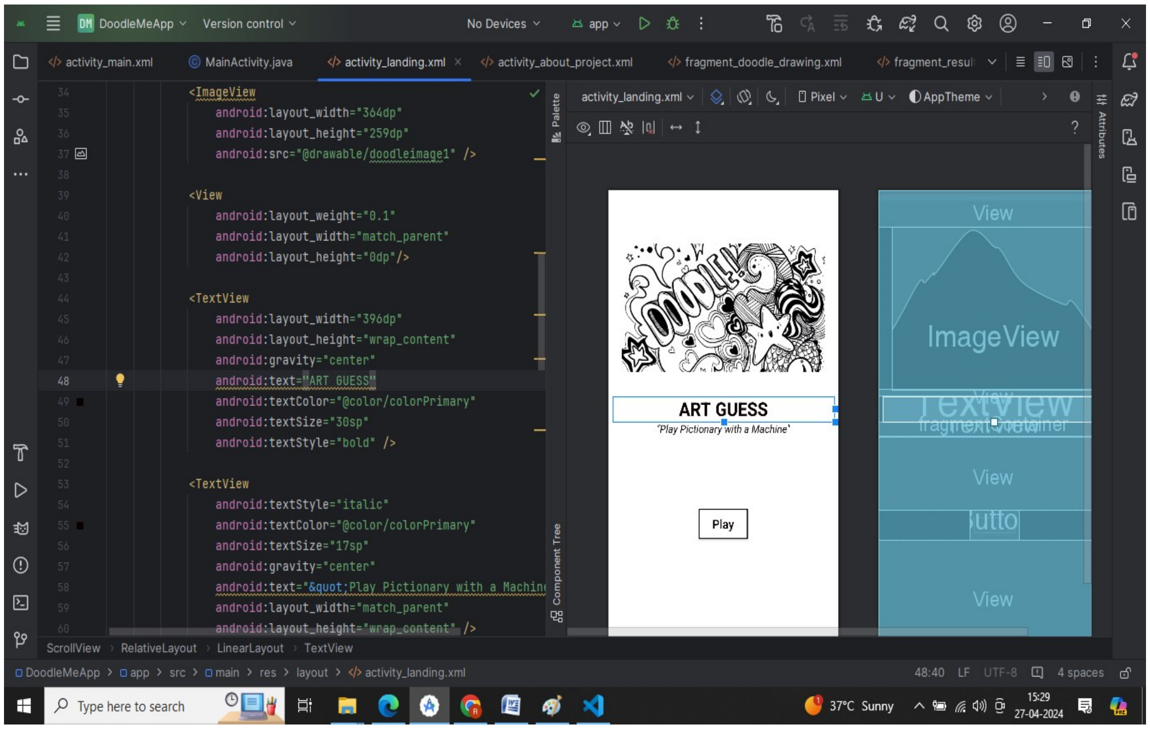
Android Application:

An Android application was developed using Android Studio, featuring a user interface for doodle drawing and interaction. The TensorFlow Lite model was integrated into the application to provide real-time prediction and feedback during drawing sessions.
The application's user interface was designed to be intuitive, with features such as word prompts, feedback display, and timer visualization enhancing the drawing experience.
The trained TensorFlow Lite model was deployed within the Android application, allowing users to draw doodles and receive instant feedback on their drawings.

CHAPTER 7

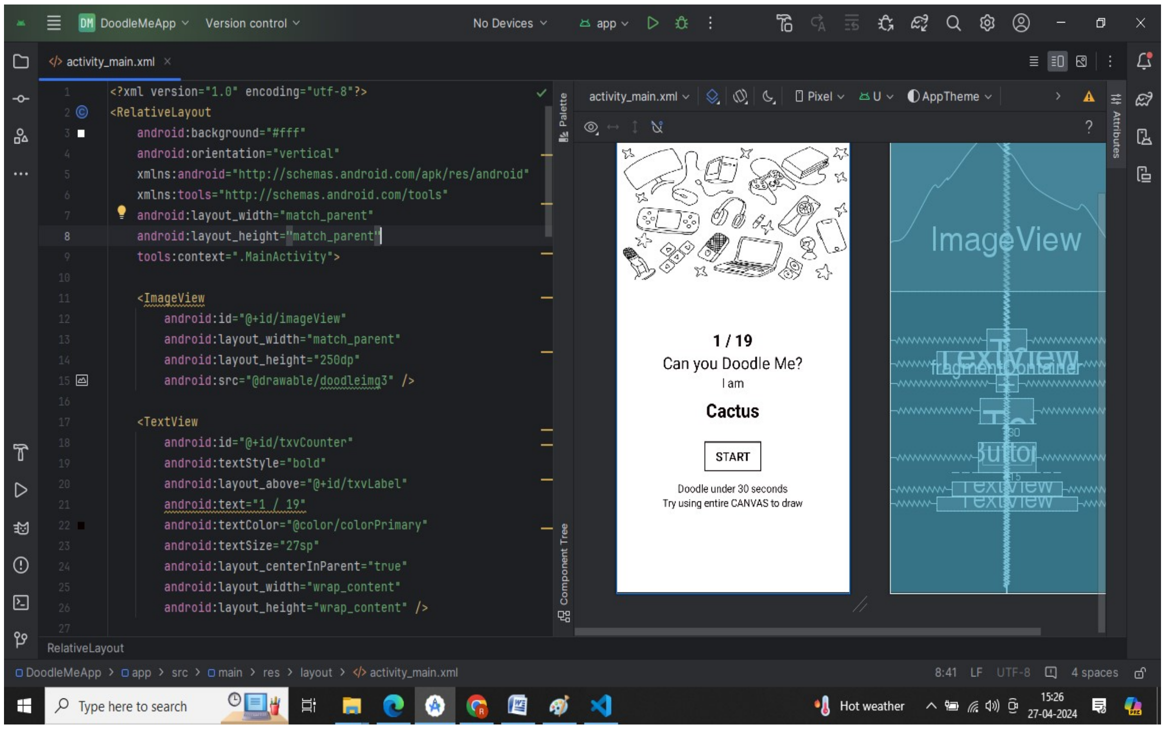
PSUEDO CODE

Activity_landing.xml page



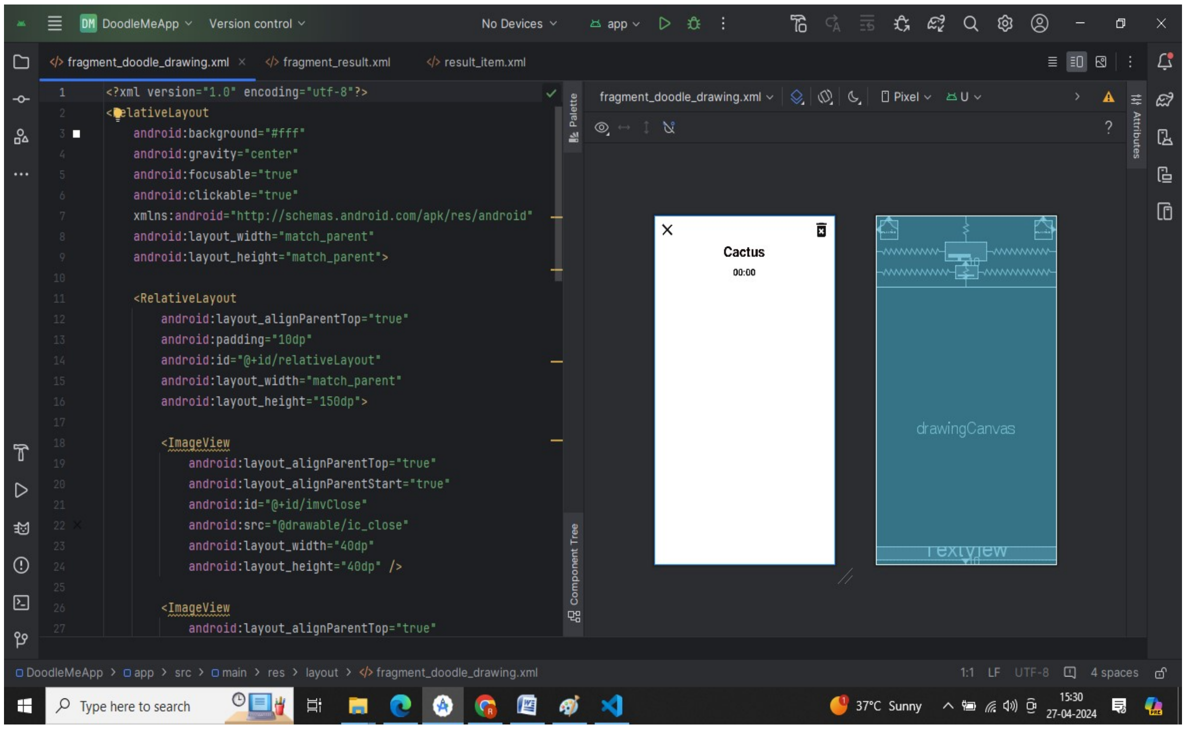
```
<ImageView
    android:layout_width="364dp"
    android:layout_height="259dp"
    android:src="@drawable/doodleimage1" />

<View
    android:layout_weight="0.1"
    android:layout_width="match_parent"
    android:layout_height="0dp"/>

<TextView
    android:layout_width="396dp"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:text="ART GUESS"
    android:textColor="@color/colorPrimary"
    android:textSize="30sp"
    android:textStyle="bold" />

<TextView
    android:textStyle="italic"
    android:textColor="@color/colorPrimary"
    android:textSize="17sp"
    android:gravity="center"
    android:text="&quot;Play Pictionary with a Machin
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

Activity_main.xml page



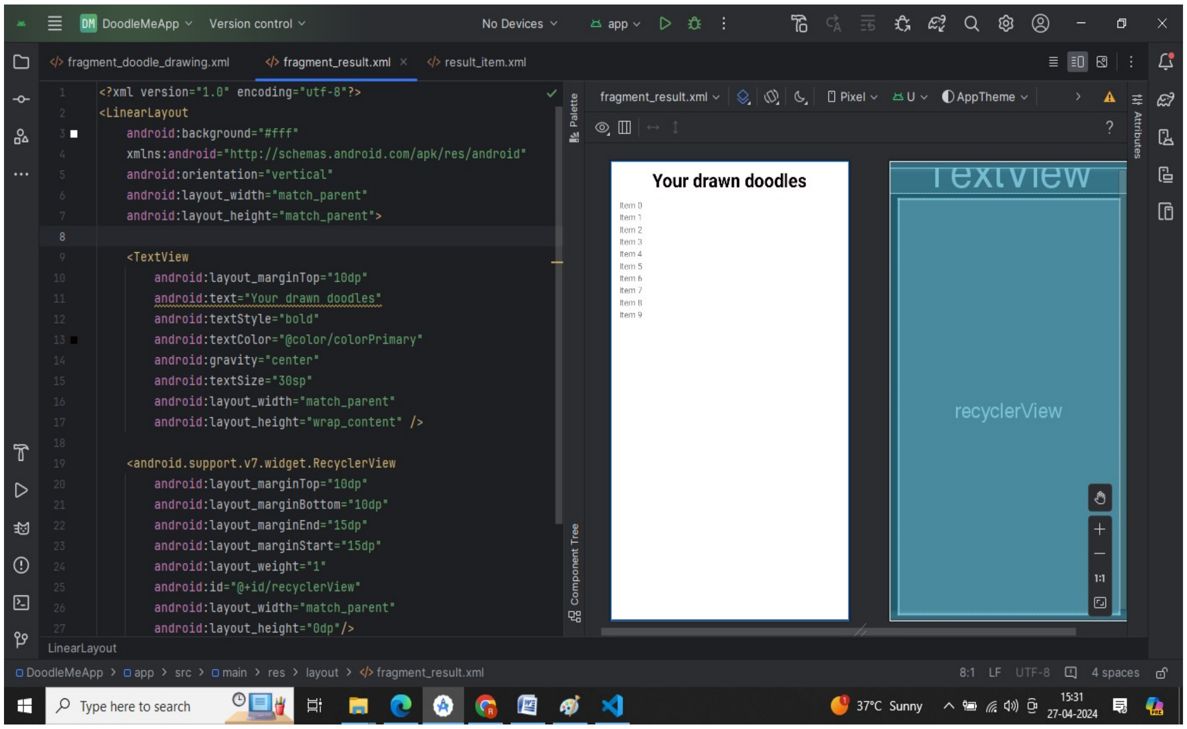
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    android:background="#fff"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="match_parent"
        android:layout_height="250dp"
        android:src="@drawable/doodleimg3" />

    <TextView
        android:id="@+id/txvCounter"
        android:textStyle="bold"
        android:layout_above="@+id/txvLabel"
        android:text="1 / 19"
        android:textColor="@color/colorPrimary"
        android:textSize="27sp"
        android:layout_centerInParent="true"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

Fragment_doodle_drawing.xml page



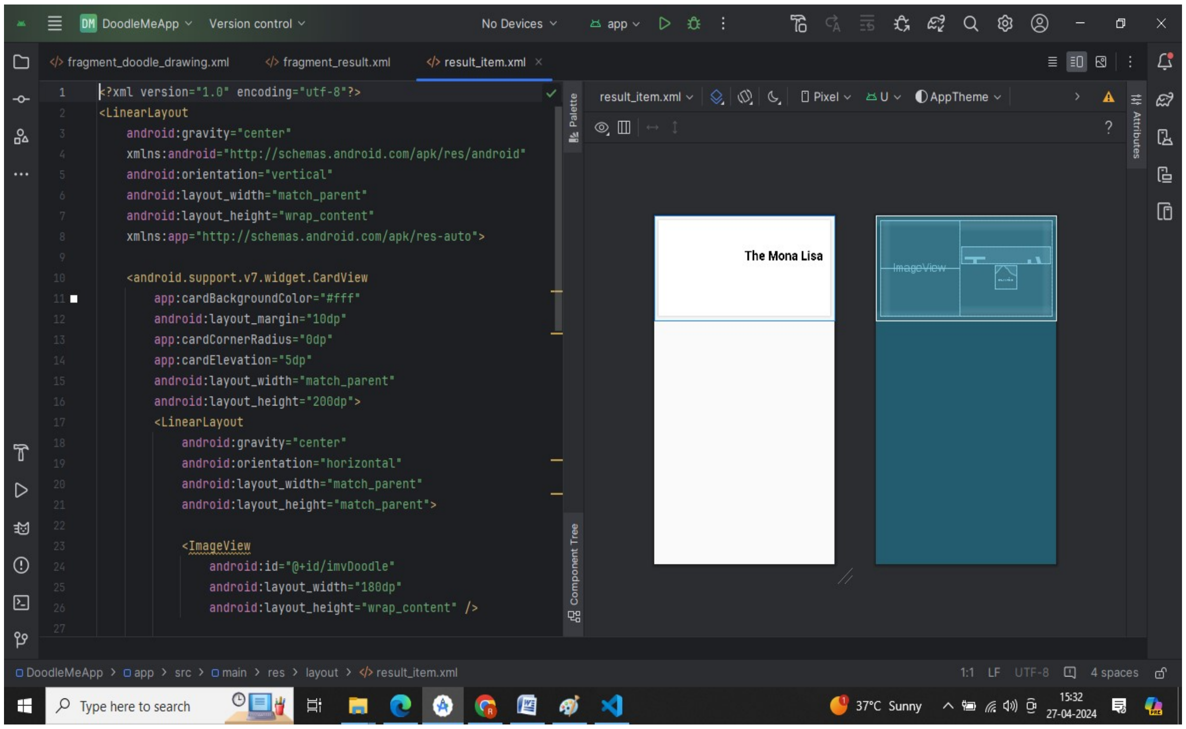
```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    android:background="#fff"
    android:gravity="center"
    android:focusable="true"
    android:clickable="true"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <RelativeLayout
        android:layout_alignParentTop="true"
        android:padding="10dp"
        android:id="@+id/relativeLayout"
        android:layout_width="match_parent"
        android:layout_height="150dp">

        <ImageView
            android:layout_alignParentTop="true"
            android:layout_alignParentStart="true"
            android:id="@+id/imvClose"
            android:src="@drawable/ic_close"
            android:layout_width="40dp"
            android:layout_height="40dp" />

        <ImageView
            android:layout_alignParentTop="true"
```

Fragment_result.xml page



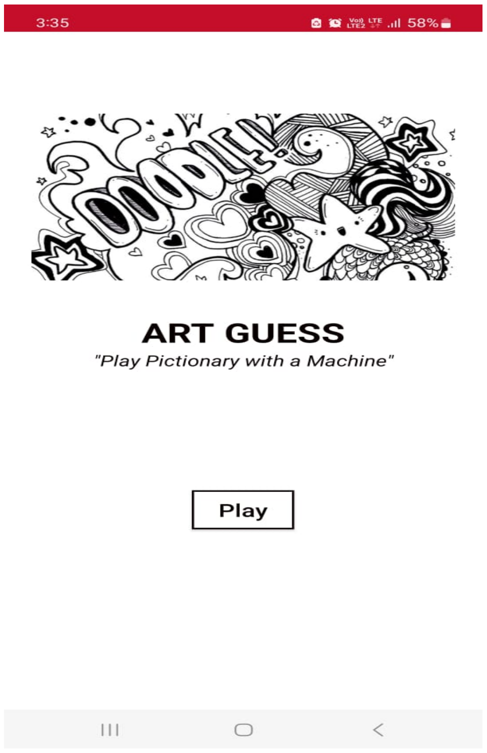
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:background="#fff"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_marginTop="10dp"
        android:text="Your drawn doodles"
        android:textStyle="bold"
        android:textColor="@color/colorPrimary"
        android:gravity="center"
        android:textSize="30sp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <android.support.v7.widget.RecyclerView
        android:layout_marginTop="10dp"
        android:layout_marginBottom="10dp"
        android:layout_marginEnd="15dp"
        android:layout_marginStart="15dp"
        android:layout_weight="1"
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="0dp"/>
```

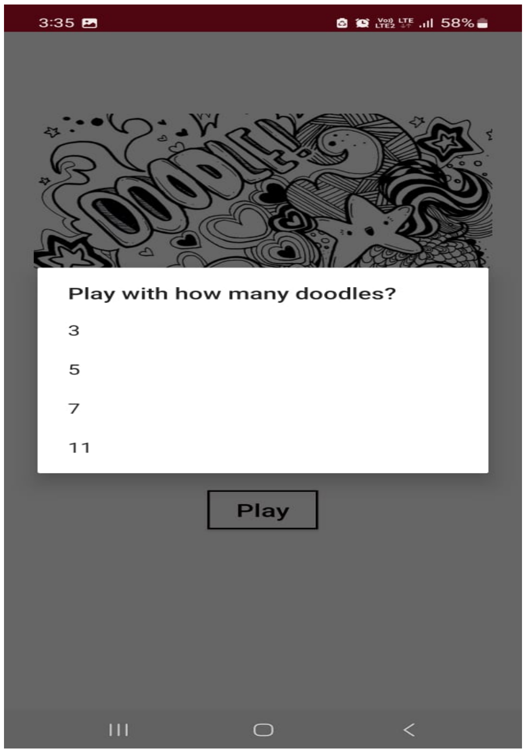
Result_item.xml

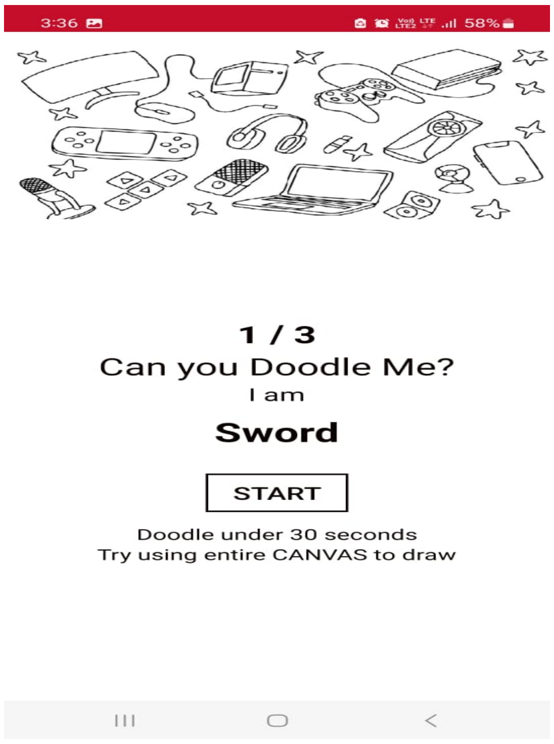


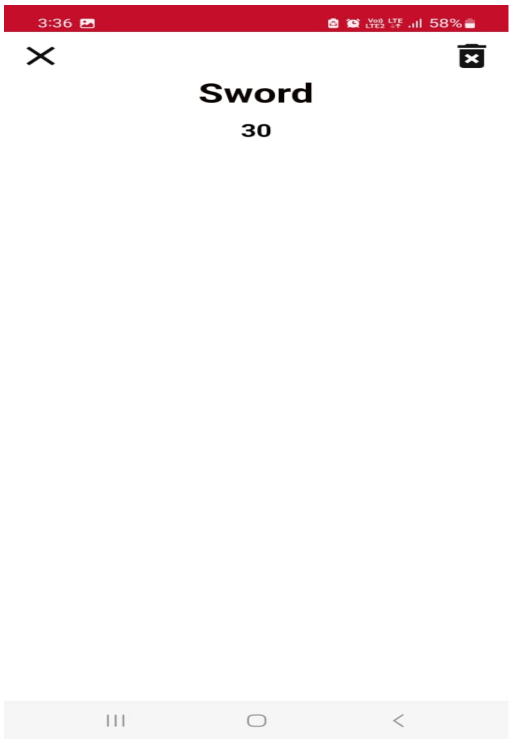
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:gravity="center"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <android.support.v7.widget.CardView
        app:cardBackgroundColor="#fff"
        android:layout_margin="10dp"
        app:cardCornerRadius="0dp"
        app:cardElevation="5dp"
        android:layout_width="match_parent"
        android:layout_height="200dp">
        <LinearLayout
            android:gravity="center"
            android:orientation="horizontal"
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <ImageView
                android:id="@+id/imvDoodle"
                android:layout_width="180dp"
                android:layout_height="wrap_content" />

```

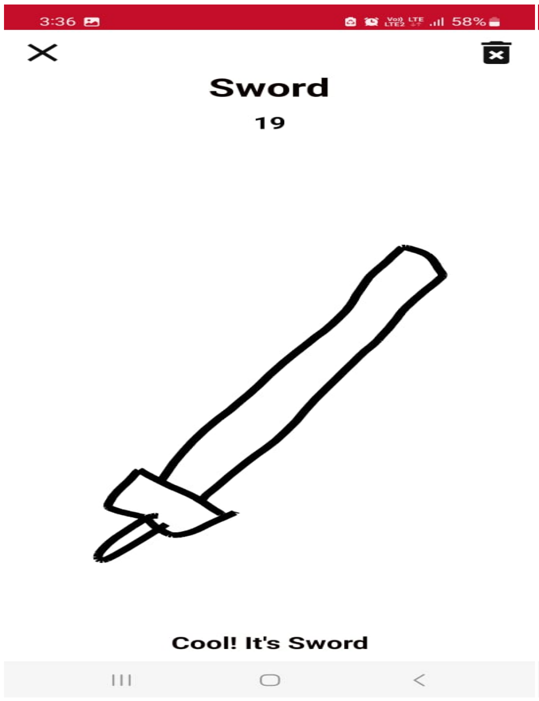
CHAPTER 8
OUTPUT SCREENS







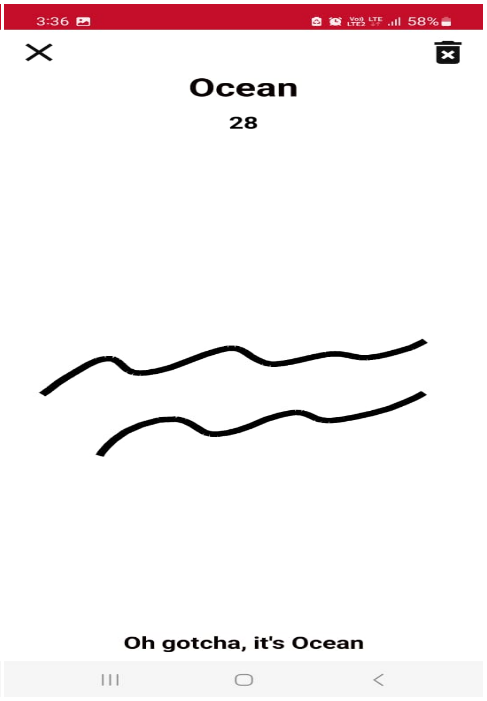




Sword

19

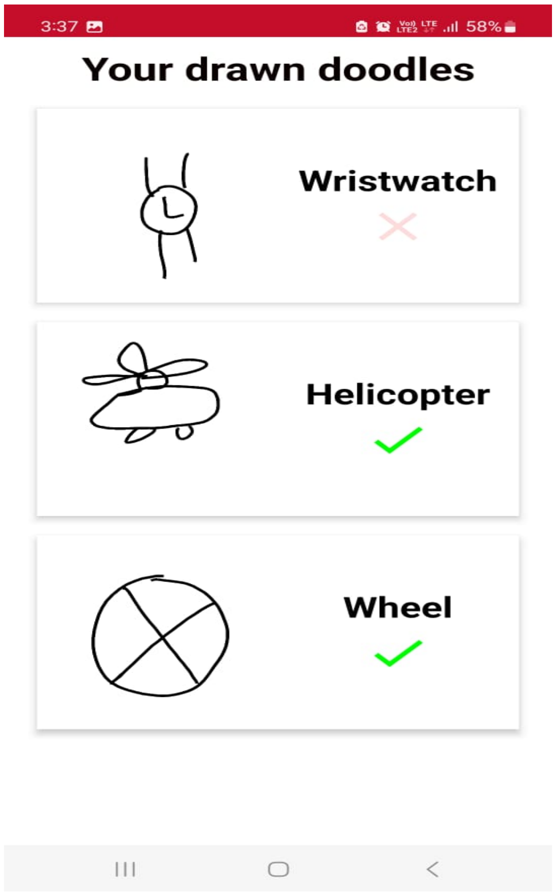
Cool! It's Sword



Ocean

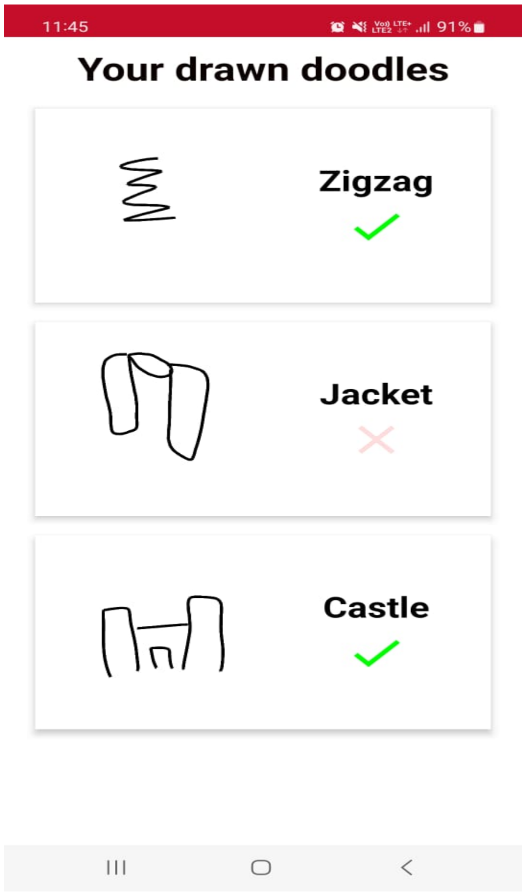
28

Oh gotcha, it's Ocean



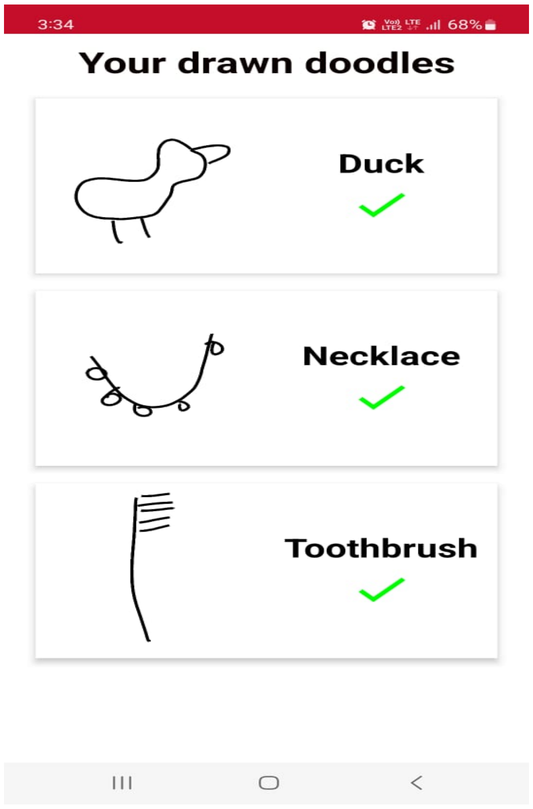
Your drawn doodles

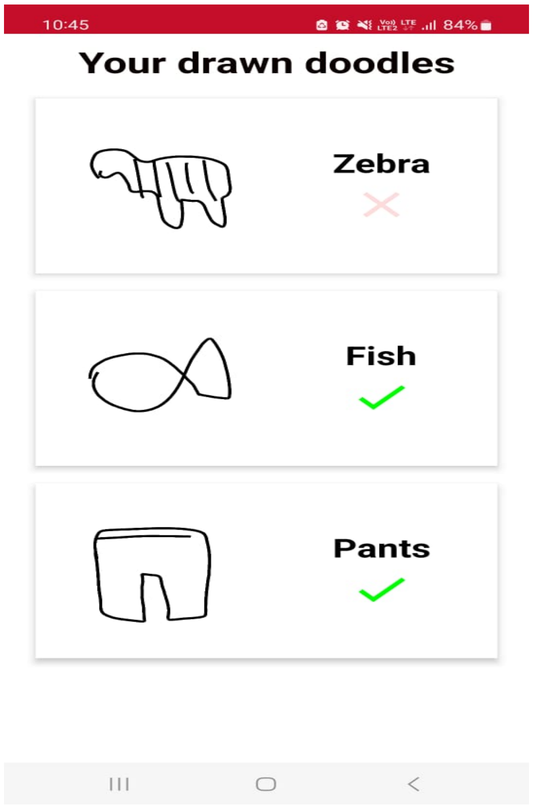
Wristwatch	✗
Helicopter	✓
Wheel	✓

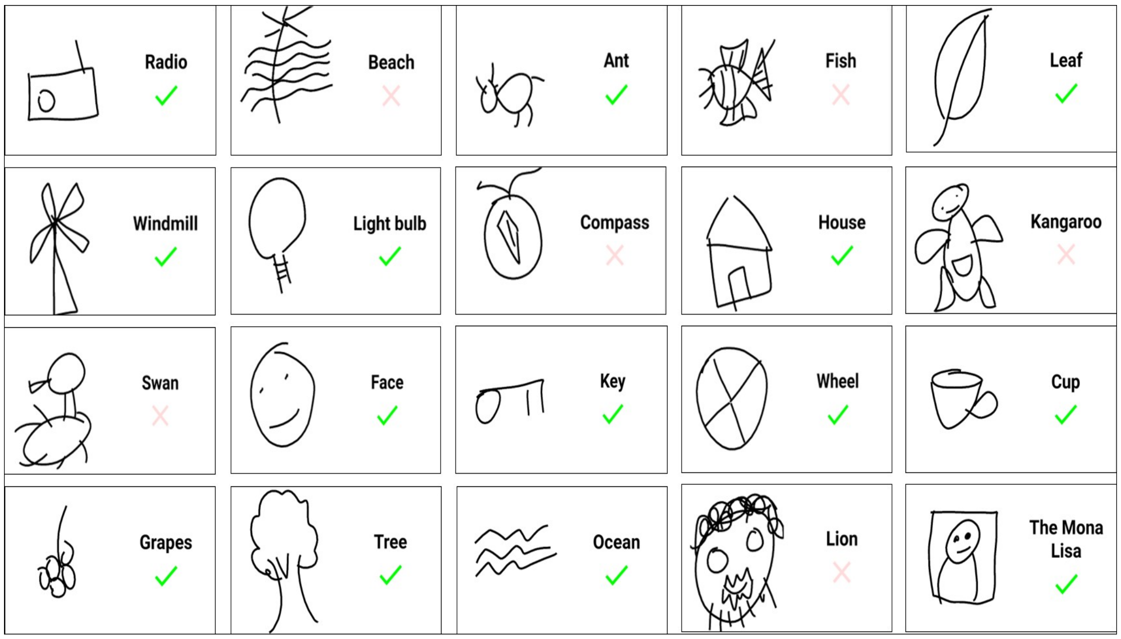


Your drawn doodles

Zigzag	✓
Jacket	✗
Castle	✓

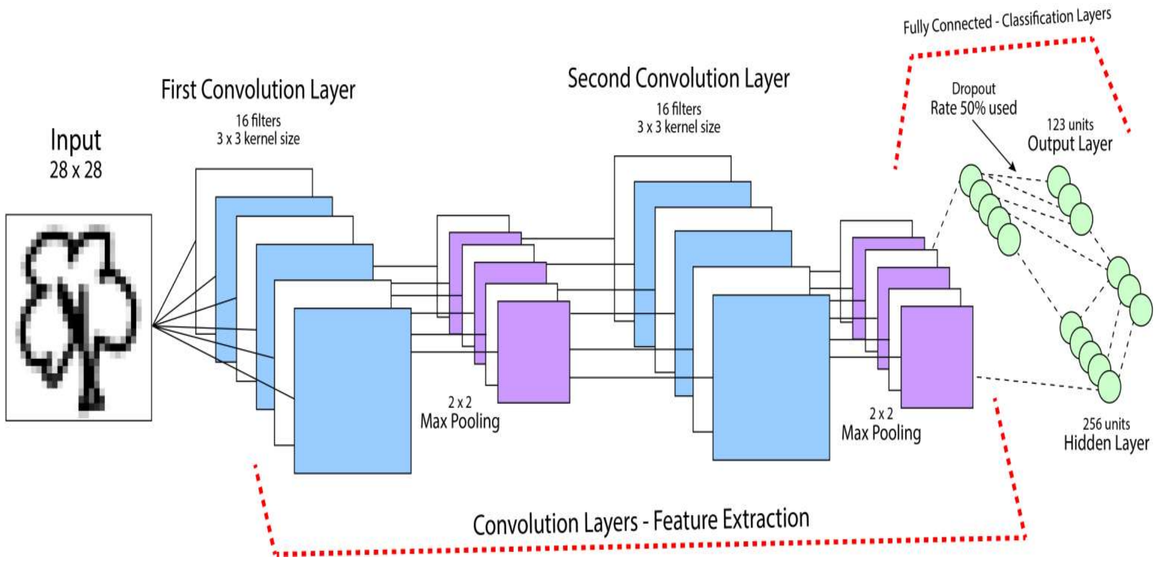

3:34
68%
Your drawn doodles
Duck ✓
Necklace ✓
Toothbrush ✓


10:45
84%
Your drawn doodles
Zebra ✗
Fish ✓
Pants ✓

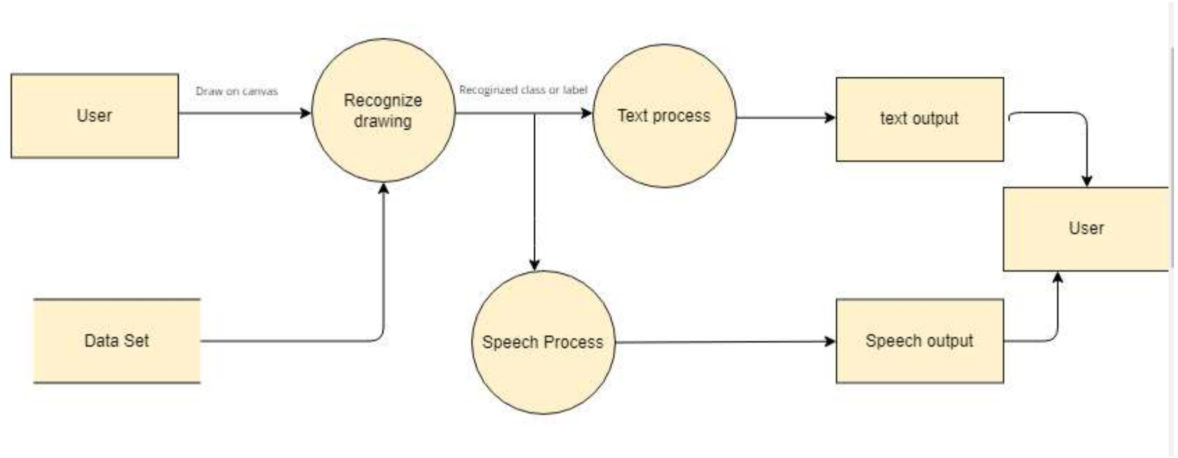

Radio ✓
Beach ✗
Ant ✓
Fish ✗
Leaf ✓
Windmill ✓
Light bulb ✓
Compass ✗
House ✓
Kangaroo ✗
Swan ✗
Face ✓
Key ✓
Wheel ✓
Cup ✓
Grapes ✓
Tree ✓
Ocean ✓
Lion ✗
The Mona Lisa ✓

CHAPTER 9
DIAGRAMS

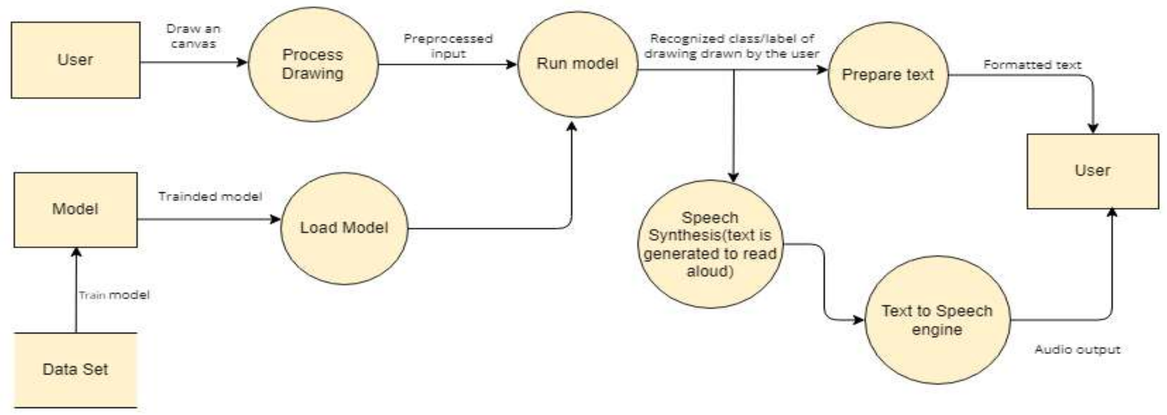
CNN Architecture Diagram



0 LEVEL DFD



1 LEVEL DFD



CHAPTER 10
TESTING

Functional Testing:

Drawing Functionality: The core functionality of the application, including the drawing canvas, word prompts, and timer, was thoroughly tested to ensure smooth operation and accurate prediction of user doodle drawings.

Feedback Mechanism: The feedback mechanism, which provided users with instant feedback on their drawings through text and speech synthesis, was tested to ensure accuracy and responsiveness.

Performance Testing:

Inference Speed: The inference speed of the TensorFlow Lite model was measured to ensure that doodle predictions were generated in real-time without significant latency.

Resource Usage: The application's resource usage, including CPU, memory, and battery consumption, was monitored during drawing sessions to identify any performance bottlenecks or resource-intensive operations.

User Experience Testing:

Usability Testing: Testers provided feedback on the user interface, drawing experience, and overall satisfaction with the application.

Error Handling: Error scenarios, such as network connectivity issues or unexpected crashes, were simulated to assess the application's error handling capabilities and resilience.

CHAPTER 11
CONCLUSIONS

In conclusion, the Art Guess project has successfully achieved its objectives of combining creativity with artificial intelligence to create an engaging and interactive drawing experience for users. Through the development of an Android application integrated with a Convolutional Neural Network (CNN) model trained on the Quick, Draw! dataset, users are able to draw doodles based on provided words and receive instant feedback on their drawings. The implementation of the project involved meticulous dataset preprocessing, model development using TensorFlow and Keras, integration with the Android application using TensorFlow Lite, and rigorous testing across various devices and scenarios. The project not only demonstrates the power and versatility of machine learning technology in enhancing user experiences but also highlights the importance of user-centric design and iterative improvement based on user feedback. Moving forward, the Art Guess application holds potential for further enhancements and expansion, including additional features, improved prediction accuracy, and broader user adoption. Overall, the project exemplifies the successful fusion of artistry and technology, offering users a fun and creative platform to express themselves through doodle drawing while showcasing the capabilities of machine learning in real-world applications.

CHAPTER 12
REFERENCES

1. https://www.tensorflow.org/lite
2. https://www.youtube.com/watch?v=AsYczuDBGp0
3. https://github.com/robo-nic/dataset
4. "A Novel Sketch Recognition Model based on Convolutional Neural Networks" Abdullah Talha Kabakus Department of Computer Engineering, Faculty of Engineering, Duzce University, Duzce, Turkey
5. https://quickdraw.withgoogle.com/data
6. https://colab.research.google.com
7. "Quick, Draw! Doodle Recognition" Kristine Guo Stanford University kguo98@stanford.edu James, WoMa Stanford University jaywoma@stanford.edu ,Eric Xu Stanford University ericxu0@stanford.edu
8. "CS231n Convolutional Neural Networks for Visual Recognition," Stanford University, 2020. https://cs231n.github.io/convolutionalnetworks (accessed Jun. 08, 2020).