LAMPIRAN

Lampiran Source Code Pembuatan Aplikasi Pengenalan Macam-Macam Hewan Dengan Android Studio

activity_main.xml | MainActivity.java | model.tflite | activity_camera.xml | ClassifierActivity.java | CameraActivity.java

```java
package org.tensorflow.lite.examples.classification;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import androidx.appcompat.app.AppCompatActivity;

import org.tensorflow.lite.examples.classification.tflite.AboutActivity;

public class MainActivity extends AppCompatActivity {


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button btn1 = findViewById(R.id.button1);
        Button btn2 = findViewById(R.id.button2);
        Button btn3 = findViewById(R.id.button3);

        btn1.setOnClickListener(v -> {
            Intent intent = new Intent( packageContext: MainActivity.this, ClassifierActivity.class);
            startActivity(intent);
        });
        btn2.setOnClickListener(v -> startActivity(new Intent(getApplicationContext(), AboutActivity.class)));

        btn3.setOnClickListener(v -> startActivity(new Intent(getApplicationContext(), HelpActivity.class)));
    }
}
```

activity_main.xml | MainActivity.java | model.tflite | activity_camera.xml | ClassifierActivity.java | Cam

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/app_bg"
    tools:context="org.tensorflow.lite.examples.classification.MainActivity">


    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/header2" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="48dp"
        android:text="Aplikasi"
        android:textColor="@android:color/black" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:orientation="vertical">

```

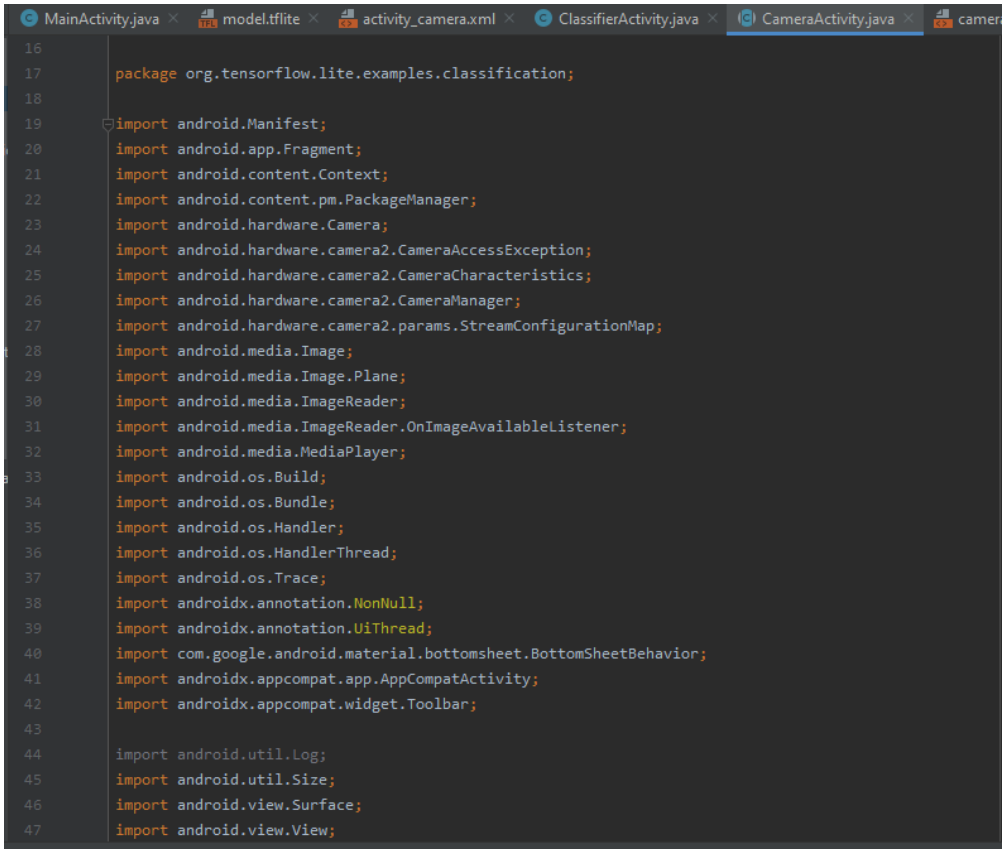
activity_main.xml | MainActivity.java | model.tflite | activity_camera.xml | ClassifierActivity.java | Cam

```

        <Button
            android:id="@+id/button1"
            android:layout_width="250dp"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:background="@drawable/button1"
            android:text="Start Scan"
            android:textAllCaps="false" />

        <Button
            android:id="@+id/button2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:background="@drawable/button1"
            android:text="About"
            android:textAllCaps="false" />

```


MainActivity.java
model.tflite
activity_camera.xml
ClassifierActivity.java
CameraActivity.java
16
17 package org.tensorflow.lite.examples.classification;
18
19 import android.Manifest;
20 import android.app.Fragment;
21 import android.content.Context;
22 import android.content.pm.PackageManager;
23 import android.hardware.Camera;
24 import android.hardware.camera2.CameraAccessException;
25 import android.hardware.camera2.CameraCharacteristics;
26 import android.hardware.camera2.CameraManager;
27 import android.hardware.camera2.params.StreamConfigurationMap;
28 import android.media.Image;
29 import android.media.Image.Plane;
30 import android.media.ImageReader;
31 import android.media.ImageReader.OnImageAvailableListener;
32 import android.media.MediaPlayer;
33 import android.os.Build;
34 import android.os.Bundle;
35 import android.os.Handler;
36 import android.os.HandlerThread;
37 import android.os.Trace;
38 import androidx.annotation.NonNull;
39 import androidx.annotation.UiThread;
40 import com.google.android.material.bottomsheet.BottomSheetBehavior;
41 import androidx.appcompat.app.AppCompatActivity;
42 import androidx.appcompat.widget.Toolbar;
43
44 import android.util.Log;
45 import android.util.Size;
46 import android.view.Surface;
47 import android.view.View;

MainActivity.java | model.tflite | activity_camera.xml | ClassifierActivity.java | CameraActivity.java | camer

```
import android.view.ViewTreeObserver;
import android.view.WindowManager;
import android.widget.AdapterView;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.MediaController;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;
import java.nio.ByteBuffer;
import java.util.List;
import org.tensorflow.lite.examples.classification.env.ImageUtils;
import org.tensorflow.lite.examples.classification.env.Logger;
import org.tensorflow.lite.examples.classification.tflite.Classifier.Device;
import org.tensorflow.lite.examples.classification.tflite.Classifier.Model;
import org.tensorflow.lite.examples.classification.tflite.Classifier.Recognition;

public abstract class CameraActivity extends AppCompatActivity
    implements OnImageAvailableListener,
        Camera.PreviewCallback,
        View.OnClickListener,
        AdapterView.OnItemSelectedListener {
  private static final Logger LOGGER = new Logger();

  private static final int PERMISSIONS_REQUEST = 1;

  private static final String PERMISSION_CAMERA = Manifest.permission.CAMERA;
  protected int previewWidth = 0;
  protected int previewHeight = 0;
  private Handler handler;
  private HandlerThread handlerThread;
  private boolean useCamera2API;
   private boolean useCamera2API;
   private boolean isProcessingFrame = false;
   private byte[][] yuvBytes = new byte[3][];
   private int[] rgbBytes = null;
   private int yRowStride;
   private Runnable postInferenceCallback;
   private Runnable imageConverter;
   private LinearLayout bottomSheetLayout;
   private LinearLayout gestureLayout;
   private BottomSheetBehavior sheetBehavior;
   protected TextView recognitionTextView,
       recognition1TextView,
       recognition2TextView,
       recognitionValueTextView,
       recognition1ValueTextView,
       recognition2ValueTextView,FinalView;
  protected TextView frameValueTextView,
       cropValueTextView,
       cameraResolutionTextView,
       rotationTextView,
       inferenceTimeTextView;
  protected ImageView bottomSheetArrowImageView;
  private ImageView plusImageView, minusImageView;
  private Spinner modelSpinner;
  private Spinner deviceSpinner;
  private TextView threadsTextView;

  private Model model = Model.FLOAT;
  private Device device = Device.CPU;
  private int numThreads = -1;

```

```
111
112     @Override
113     protected void onCreate(final Bundle savedInstanceState) {
114       LOGGER.d("onCreate " + this);
115       super.onCreate( savedInstanceState: null);
116       getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
117
118       setContentView(R.layout.activity_camera);
119       Toolbar toolbar = findViewById(R.id.toolbar);
120       setSupportActionBar(toolbar);
121       getSupportActionBar().setDisplayShowTitleEnabled(false);
122
123       if (hasPermission()) {
124         setFragment();
125       } else {
126         requestPermission();
127       }
128
129       threadsTextView = findViewById(R.id.threads);
130       plusImageView = findViewById(R.id.plus);
131       minusImageView = findViewById(R.id.minus);
132       modelSpinner = findViewById(R.id.model_spinner);
133       deviceSpinner = findViewById(R.id.device_spinner);
134       bottomSheetLayout = findViewById(R.id.bottom_sheet_layout);
135       gestureLayout = findViewById(R.id.gesture_layout);
136       sheetBehavior = BottomSheetBehavior.from(bottomSheetLayout);
137       bottomSheetArrowImageView = findViewById(R.id.bottom_sheet_arrow);
138
139       ViewTreeObserver vto = gestureLayout.getViewTreeObserver();
140       vto.addOnGlobalLayoutListener(
141           new ViewTreeObserver.OnGlobalLayoutListener() {
142             @Override
```

```
49          <Button
50              android:id="@+id/button3"
51              android:layout_width="match_parent"
52              android:layout_height="wrap_content"
53              android:layout_gravity="center"
54              android:background="@drawable/button1"
55              android:text="Help"
56              android:textAllCaps="false" />
57
58      </LinearLayout>
59
60
61  </RelativeLayout>
```

```
        @Override
        public void onGlobalLayout() {
          if (Build.VERSION.SDK_INT < Build.VERSION_CODES.JELLY_BEAN) {
            gestureLayout.getViewTreeObserver().removeGlobalOnLayoutListener( victim: this);
          } else {
            gestureLayout.getViewTreeObserver().removeOnGlobalLayoutListener( victim: this);
          }
          //               int width = bottomSheetLayout.getMeasuredWidth();
          //int height = gestureLayout.getMeasuredHeight();

          sheetBehavior.setPeekHeight(0);
        }
      });
    sheetBehavior.setHideable(false);

    sheetBehavior.setBottomSheetCallback(
        new BottomSheetBehavior.BottomSheetCallback() {
          @Override
          public void onStateChanged(@NonNull View bottomSheet, int newState) {
            switch (newState) {
              case BottomSheetBehavior.STATE_HIDDEN:
                break;
              case BottomSheetBehavior.STATE_EXPANDED:
                {
                  bottomSheetArrowImageView.setImageResource(R.drawable.icn_chevron_down);
                }
                break;
              case BottomSheetBehavior.STATE_COLLAPSED:
                {
                  bottomSheetArrowImageView.setImageResource(R.drawable.icn_chevron_up);
                }
                break;
              case BottomSheetBehavior.STATE_DRAGGING:
                break;
              case BottomSheetBehavior.STATE_SETTLING:
                bottomSheetArrowImageView.setImageResource(R.drawable.icn_chevron_up);
                break;
            }
          }

          @Override
          public void onSlide(@NonNull View bottomSheet, float slideOffset) {}
        });

    recognitionTextView = findViewById(R.id.detected_item);
    FinalView =findViewById(R.id.Final);
    recognitionValueTextView = findViewById(R.id.detected_item_value);
    recognition1TextView = findViewById(R.id.detected_item1);
    recognition1ValueTextView = findViewById(R.id.detected_item1_value);
    recognition2TextView = findViewById(R.id.detected_item2);
    recognition2ValueTextView = findViewById(R.id.detected_item2_value);

    frameValueTextView = findViewById(R.id.frame_info);
    cropValueTextView = findViewById(R.id.crop_info);
    cameraResolutionTextView = findViewById(R.id.view_info);
    rotationTextView = findViewById(R.id.rotation_info);
    inferenceTimeTextView = findViewById(R.id.inference_info);

    modelSpinner.setOnItemSelectedListener(this);
    deviceSpinner.setOnItemSelectedListener(this);
```

```
    plusImageView.setOnClickListener(this);
    minusImageView.setOnClickListener(this);

    model = Model.valueOf(modelSpinner.getSelectedItem().toString().toUpperCase());
    device = Device.valueOf(deviceSpinner.getSelectedItem().toString());
    numThreads = Integer.parseInt(threadsTextView.getText().toString().trim());
  }

  protected int[] getRgbBytes() {
    imageConverter.run();
    return rgbBytes;
  }

  protected int getLuminanceStride() { return yRowStride; }

  protected byte[] getLuminance() { return yuvBytes[0]; }

  /** Callback for android.hardware.Camera API */
  @Override
  public void onPreviewFrame(final byte[] bytes, final Camera camera) {
    if (isProcessingFrame) {
      LOGGER.w("Dropping frame!");
      return;
    }

    try {
      // Initialize the storage bitmaps once when the resolution is known.
      if (rgbBytes == null) {
        Camera.Size previewSize = camera.getParameters().getPreviewSize();
        previewHeight = previewSize.height;
        previewWidth = previewSize.width;
        rgbBytes = new int[previewWidth * previewHeight];
        onPreviewSizeChosen(new Size(previewSize.width, previewSize.height), rotation: 90);
      }
    } catch (final Exception e) {
      LOGGER.e(e, format: "Exception!");
      return;
    }

    isProcessingFrame = true;
    yuvBytes[0] = bytes;
    yRowStride = previewWidth;

    imageConverter =
        new Runnable() {
          @Override
          public void run() {
            ImageUtils.convertYUV420SPToARGB8888(bytes, previewWidth, previewHeight, rgbBytes);
          }
        };

    postInferenceCallback =
        new Runnable() {
          @Override
          public void run() {
            camera.addCallbackBuffer(bytes);
            isProcessingFrame = false;
          }
        };
```

```
266         processImage();
267       }
268
269       /** Callback for Camera2 API */
270       @Override
271       public void onImageAvailable(final ImageReader reader) {
272         // We need wait until we have some size from onPreviewSizeChosen
273         if (previewWidth == 0 || previewHeight == 0) {
274           return;
275         }
276         if (rgbBytes == null) {
277           rgbBytes = new int[previewWidth * previewHeight];
278         }
279         try {
280           final Image image = reader.acquireLatestImage();
281
282           if (image == null) {
283             return;
284           }
285
286           if (isProcessingFrame) {
287             image.close();
288             return;
289           }
290           isProcessingFrame = true;
291           Trace.beginSection( sectionName: "imageAvailable");
292           final Plane[] planes = image.getPlanes();
293           fillBytes(planes, yuvBytes);
294           yRowStride = planes[0].getRowStride();
295           final int uvRowStride = planes[1].getRowStride();
296           final int uvPixelStride = planes[1].getPixelStride();
```

```
      imageConverter =
          new Runnable() {
            @Override
            public void run() {
              ImageUtils.convertYUV420ToARGB8888(
                  yuvBytes[0],
                  yuvBytes[1],
                  yuvBytes[2],
                  previewWidth,
                  previewHeight,
                  yRowStride,
                  uvRowStride,
                  uvPixelStride,
                  rgbBytes);
            }
          };

      postInferenceCallback =
          new Runnable() {
            @Override
            public void run() {
              image.close();
              isProcessingFrame = false;
            }
          };

      processImage();
    } catch (final Exception e) {
      LOGGER.e(e, format: "Exception!");
      Trace.endSection();
      return;
    }
    Trace.endSection();
  }

  @Override
  public synchronized void onStart() {
    LOGGER.d("onStart " + this);
    super.onStart();
  }

  @Override
  public synchronized void onResume() {
    LOGGER.d("onResume " + this);
    super.onResume();

    handlerThread = new HandlerThread( name: "inference");
    handlerThread.start();
    handler = new Handler(handlerThread.getLooper());
  }

  @Override
  public synchronized void onPause() {
    LOGGER.d("onPause " + this);

    handlerThread.quitSafely();
    try {
      handlerThread.join();
      handlerThread = null;
```

```
      handler = null;
    } catch (final InterruptedException e) {
      LOGGER.e(e, format: "Exception!");
    }

    super.onPause();
  }

  @Override
  public synchronized void onStop() {
    LOGGER.d("onStop " + this);
    super.onStop();
  }

  @Override
  public synchronized void onDestroy() {
    LOGGER.d("onDestroy " + this);
    super.onDestroy();
  }

  protected synchronized void runInBackground(final Runnable r) {
    if (handler != null) {
      handler.post(r);
    }
  }

  @Override
  public void onRequestPermissionsResult(
      final int requestCode, final String[] permissions, final int[] grantResults) {
    if (requestCode == PERMISSIONS_REQUEST) {
      if (grantResults.length > 0
          && grantResults[0] == PackageManager.PERMISSION_GRANTED
          && grantResults[1] == PackageManager.PERMISSION_GRANTED) {
        setFragment();
      } else {
        requestPermission();
      }
    }
  }

  private boolean hasPermission() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
      return checkSelfPermission(PERMISSION_CAMERA) == PackageManager.PERMISSION_GRANTED;
    } else {
      return true;
    }
  }

  private void requestPermission() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
      if (shouldShowRequestPermissionRationale(PERMISSION_CAMERA)) {
        Toast.makeText(
                context: CameraActivity.this,
                text: "Camera permission is required for this app",
                Toast.LENGTH_LONG)
            .show();
      }
      requestPermissions(new String[] {PERMISSION_CAMERA}, PERMISSIONS_REQUEST);
    }
  }
```

```
417
418     // Returns true if the device supports the required hardware level, or better.
419 @   private boolean isHardwareLevelSupported(
420         CameraCharacteristics characteristics, int requiredLevel) {
421       int deviceLevel = characteristics.get(CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL);
422       if (deviceLevel == CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL_LEGACY) {
423         return requiredLevel == deviceLevel;
424       }
425       // deviceLevel is not LEGACY, can use numerical sort
426       return requiredLevel <= deviceLevel;
427     }
428
429 @   private String chooseCamera() {
430       final CameraManager manager = (CameraManager) getSystemService(Context.CAMERA_SERVICE);
431       try {
432         for (final String cameraId : manager.getCameraIdList()) {
433           final CameraCharacteristics characteristics = manager.getCameraCharacteristics(cameraId);
434
435           // We don't use a front facing camera in this sample.
436           final Integer facing = characteristics.get(CameraCharacteristics.LENS_FACING);
437           if (facing != null && facing == CameraCharacteristics.LENS_FACING_FRONT) {
438             continue;
439           }
440
441           final StreamConfigurationMap map =
442               characteristics.get(CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);
443
444           if (map == null) {
445             continue;
446           }
447
448           // Fallback to camera1 API for internal cameras that don't have full support.
449           // This should help with legacy situations where using the camera2 API causes
450           // distorted or otherwise broken previews.
451           useCamera2API =
452               (facing == CameraCharacteristics.LENS_FACING_EXTERNAL)
453                   || isHardwareLevelSupported(
454                       characteristics, CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL_FULL);
455           LOGGER.i("Camera API lv2?: %s", useCamera2API);
456           return cameraId;
457         }
458       } catch (CameraAccessException e) {
459         LOGGER.e(e,  format: "Not allowed to access camera");
460       }
461
462       return null;
463     }
464
465     protected void setFragment() {
466       String cameraId = chooseCamera();
467
468       Fragment fragment;
469       if (useCamera2API) {
470         CameraConnectionFragment camera2Fragment =
471             CameraConnectionFragment.newInstance(
472                 new CameraConnectionFragment.ConnectionCallback() {
473                   @Override
474 o↑                public void onPreviewSizeChosen(final Size size, final int rotation) {
475                     previewHeight = size.getHeight();
476                     previewWidth = size.getWidth();
```

```
                    previewWidth = size.getWidth();
                    CameraActivity.this.onPreviewSizeChosen(size, rotation);
                  }
                },
                imageListener: this,
                getLayoutId(),
                getDesiredPreviewFrameSize());

        camera2Fragment.setCamera(cameraId);
        fragment = camera2Fragment;
      } else {
        fragment =
            new LegacyCameraConnectionFragment( imageListener: this, getLayoutId(), getDesiredPreviewFrameSize());
      }

      getFragmentManager().beginTransaction().replace(R.id.container, fragment).commit();
    }

    protected void fillBytes(final Plane[] planes, final byte[][] yuvBytes) {
      // Because of the variable row stride it's not possible to know in
      // advance the actual necessary dimensions of the yuv planes.
      for (int i = 0; i < planes.length; ++i) {
        final ByteBuffer buffer = planes[i].getBuffer();
        if (yuvBytes[i] == null) {
          LOGGER.d("Initializing buffer %d at size %d", i, buffer.capacity());
          yuvBytes[i] = new byte[buffer.capacity()];
        }
        buffer.get(yuvBytes[i]);
      }
    }

    protected void readyForNextImage() {
      if (postInferenceCallback != null) {
        postInferenceCallback.run();
      }
    }

    protected int getScreenOrientation() {
      switch (getWindowManager().getDefaultDisplay().getRotation()) {
        case Surface.ROTATION_270:
          return 270;
        case Surface.ROTATION_180:
          return 180;
        case Surface.ROTATION_90:
          return 90;
        default:
          return 0;
      }
    }
```

```
@UiThread
protected void showResultsInBottomSheet(List<Recognition> results) {









 if (results != null && results.size() >= 3) {
   Recognition recognition = results.get(0);
   if (recognition != null) {
     if (recognition.getTitle() != null){

       if (recognition.getConfidence() != null && (recognition.getConfidence()) <= 0.95){
         FinalView.setText("Hewan Tidak Ditemukan");
       }else if (recognition.getConfidence() != null && (recognition.getConfidence()) >= 0.98){
         // the final result will be displayed from here.
         String s1=recognition.getTitle();
         if(s1.equals("0_Singa"))
         {

           FinalView.setText("Ini Singa ");
           final MediaPlayer mp100 = MediaPlayer.create( context: this,R.raw.singa);
           TextView tv = findViewById(R.id.Final);
           tv.setOnClickListener(new View.OnClickListener() {
             @Override
             public void onClick(View v) { mp100.start(); }
           });









         }
         else if(s1.equals("1_Beruang") )
         {
           FinalView.setText("Ini Beruang ");
           final MediaPlayer mp10 = MediaPlayer.create( context: this,R.raw.beruang);
           TextView tv = findViewById(R.id.Final);
           tv.setOnClickListener(new View.OnClickListener() {
             @Override
             public void onClick(View v) { mp10.start(); }
           });

         }
         else if(s1.equals("2_Jerapah"))
         {
           FinalView.setText("Ini Jerapah ");
           TextView tv = findViewById(R.id.Final);
           final MediaPlayer mp2000 = MediaPlayer.create( context: this,R.raw.jerapah);

           tv.setOnClickListener(new View.OnClickListener() {
             @Override
             public void onClick(View v) { mp2000.start(); }
           });

         }
         else {
           FinalView.setText("Data tidak ditemukan");
         }
       }
```


Ini Orang Utan
Ini Orang Utan
Ini Orang Utan
Ini Orang Utan
Ini Orang Utan


Ini Badak
Ini Badak
Hewan Tidak Ditemukan
Ini Badak
Ini Badak


Ini Burung Hantu
Ini Burung Hantu
Ini Burung Hantu
Ini Burung Hantu
Ini Burung Hantu