```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array, save_img
import ipywidgets as widgets
from IPython.display import display

def generate_synthetic_data():
    if not os.path.exists('synthetic_data/train'):
        os.makedirs('synthetic_data/train/labrador')
        os.makedirs('synthetic_data/train/poodle')
        os.makedirs('synthetic_data/train/bulldog')
        os.makedirs('synthetic_data/validation/labrador')
        os.makedirs('synthetic_data/validation/poodle')
        os.makedirs('synthetic_data/validation/bulldog')

    breeds = ['labrador', 'poodle', 'bulldog']

    for breed in breeds:
        for i in range(50):  # 50 images per breed for training
            img = np.ones((128, 128, 3)) * np.random.rand(1, 1, 3)  # Random color
            save_img(f'synthetic_data/train/{breed}/{breed}_{i}.jpg', img)

        for i in range(10):  # 10 images per breed for validation
            img = np.ones((128, 128, 3)) * np.random.rand(1, 1, 3)
            save_img(f'synthetic_data/validation/{breed}/{breed}_{i}.jpg', img)

generate_synthetic_data()

# Define the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(3, activation='softmax')  # 3 breeds for synthetic data
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Data augmentation for training
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

validation_datagen = ImageDataGenerator(rescale=1./255)

# Data generators for training and validation
train_generator = train_datagen.flow_from_directory(
    'synthetic_data/train',
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical'
```

```python
)

validation_generator = validation_datagen.flow_from_directory(
    'synthetic_data/validation',
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical'
)

# Train the model
model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size,
    epochs=5  # Use fewer epochs for faster training on synthetic data
)

# Save the trained model
model.save('synthetic_dog_breed_model.h5')

# Function to predict dog breed after image upload
def predict_dog_breed(model_path='synthetic_dog_breed_model.h5'):
    model = tf.keras.models.load_model(model_path)

    # Create a file upload widget
    uploader = widgets.FileUpload(
        accept='image/*',  # Accept only image files
        multiple=False  # Allow only one file upload
    )
    display(uploader)

    # Handle the uploaded file
    def on_upload_change(change):
        if change['type'] == 'change' and change['name'] == 'value' and len(uploader.value) > 0:
            uploaded_file = list(uploader.value.values())[0]  # Get the uploaded file
            content = uploaded_file['content']  # Get the content of the uploaded file
            with open('temp_image.jpg', 'wb') as f:  # Save content to a temporary file
                f.write(content)

            # Load the uploaded image
            img = load_img('temp_image.jpg', target_size=(128, 128))
            img_array = img_to_array(img) / 255.0  # Preprocess the image
            img_array = np.expand_dims(img_array, axis=0)

            # Predict the breed
            prediction = model.predict(img_array)
            breed_index = np.argmax(prediction)
            breed_labels = train_generator.class_indices
            breed_name = list(breed_labels.keys())[list(breed_labels.values()).index(breed_index)]

            # Display the image and predicted breed
            plt.imshow(img)
            plt.title(f'Predicted breed: {breed_name}')
            plt.axis('off')
            plt.show()

    uploader.observe(on_upload_change)  # Observe the upload event

# Call the prediction function
predict_dog_breed()
```

```
Found 150 images belonging to 3 classes.
Found 30 images belonging to 3 classes.
Epoch 1/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 870ms/step - accuracy: 0.2667 - loss: 1.2690 - val_accuracy: 0.3333 - val_loss: 1
Epoch 2/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 92ms/step - accuracy: 0.2188 - loss: 1.1654 - val_accuracy: 0.2000 - val_loss: 1.
Epoch 3/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.3229 - loss: 1.1096 - val_accuracy: 0.3333 - val_loss: 1.10
Epoch 4/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 215ms/step - accuracy: 0.1364 - loss: 1.1641 - val_accuracy: 0.3333 - val_loss: 1
Epoch 5/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 886ms/step - accuracy: 0.3594 - loss: 1.0932 - val_accuracy: 0.2667 - val_loss: 1
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics
```

Upload (1)

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step
```

Predicted breed: poodle