Dog Breeds Identification Documentation

Introduction

- This application is able to recognize 120 dog breeds, using your phone camera and Neural Networks algorithms (MobileNetV2). It also remembers the dogs that you recognized, in a History-type screen.
- We made it to improve the basic human knowledge about the many different dog breeds. It offers a pleasant user experience through a simple User Interface, and a very quick breed recognition.

Design and implementation

The implementation is divided in two sections:

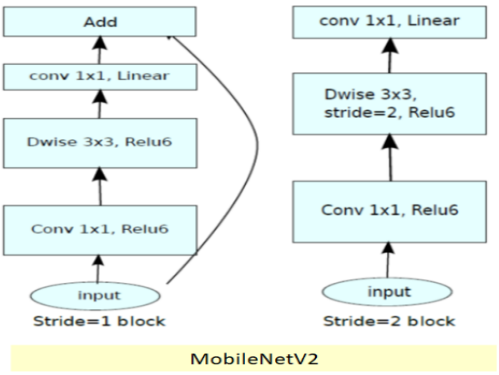
Neural Network Implementation:

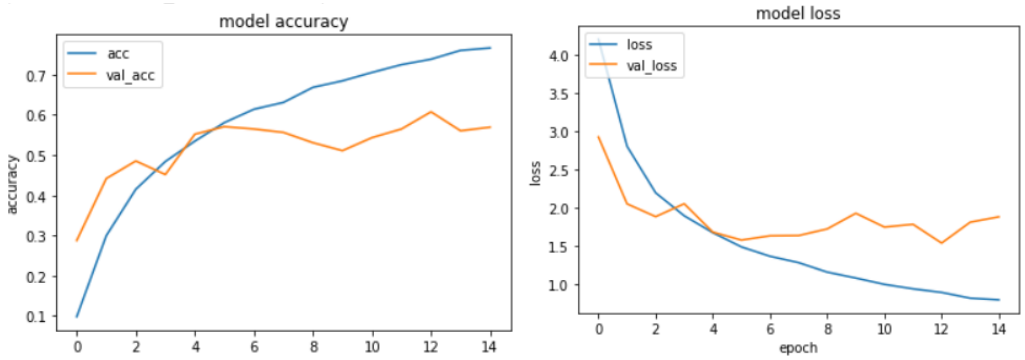
- To be able to recognize different dog breeds, our application uses a custom made NN. We used google colab as working platform.
- First, our model is trained on a dataset provided by Kaggle[1], that consist of 9982 pictures. This dataset is split in 7035 training images and 2947 validation images. The images are split into 120 folders, one for each breed, and are labeled correctly. We used a Batch size of 16, and trained for 15 epochs. The image size will be 224.
- ImageDataGenerator objects are used to rescale all the images to the correct size, and to use real-time data augmentation, which basically generates new images from the original ones, so the training set is larger:
- Next step was to create a model. As the base model, we tested a few pre-trained networks, all were using the ImageNet weights. The tested networks were: MobileNetV2, ResNet50 and InceptionV3.

```
base_model = MobileNetV2(weights='imagenet',
include_top=False, pooling='avg',
input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3))

x = base_model.output
x = Dense(512)(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)

predictions = Dense(NUM_CLASSES,
activation='softmax')(x)
```

```
train_datagen = ImageDataGenerator(
  preprocessing_function=preprocess,
  #rescale=1./255, # done in preprocess()
  # randomly rotate images (degrees, 0 to 30)
  rotation_range=30,
  # randomly shift images horizontally
  # (fraction of total width)
  width_shift_range=0.3,
  height_shift_range=0.3,
  # randomly flip images
  horizontal_flip=True,
  vertical_flip=False,
  zoom_range=0.3)
```

- The last one had the best accuracy scores, but there was a problem with its output values, so we went with the MobileNetV2. Over the base model, we added four more layers, for our specific output.
- We froze the weights of the base model, and trained it using as optimizer ‘sgd’, and as loss function, ‘categorical_crossentropy’.
- The model starts to overfit after 7 epochs, so the validation accuracy stops improving after that.
- The model hits an accuracy of 76,63% on the training set and 60,76% on the validation set.
- The accuracy and loss plots are looking like this:





- After saving our model, we converted it to a .tflite file, so it can be used in our application.

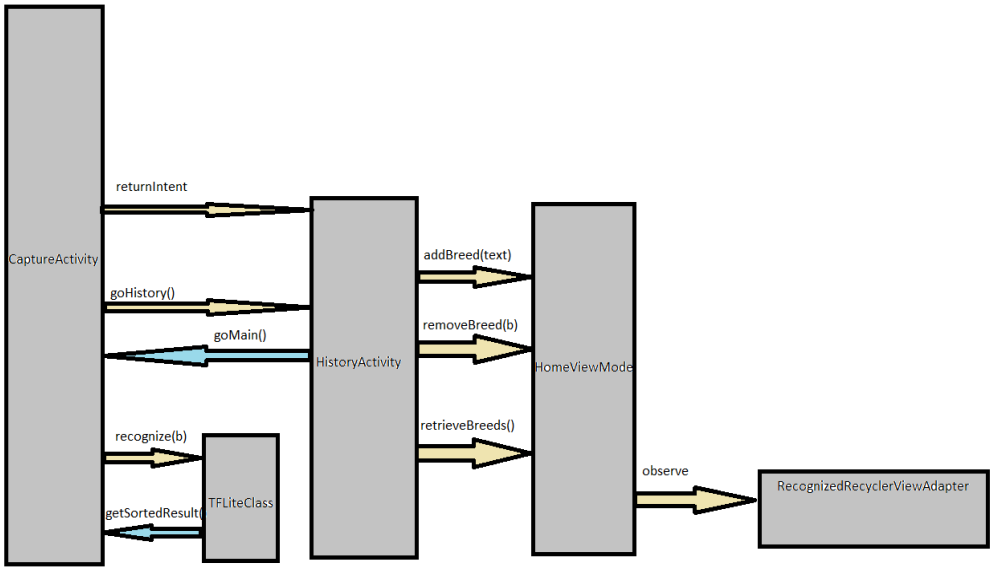
Android Implementation:

Our application implements three main subjects from the lab:

1. **TensorFlowLite**
2. **RecyclerView**
3. **Room**

- First, using a CameraView from the main activity, an image is captured as a bitmap, and fed into a classifier, that’s using the model that we have created previously.
- After the classifier is created, and it has the model, and the labels already loaded, it converts the image from bitmap, to a ByteBuffer, that is using floats instead of bytes, because that’s what our model requires. Also, when converting the bitmap, we use an image mean and an image standard deviation of 128.
- After the conversion, we run the Interpreter object that is using our model, on the ByteBuffer, expecting an array of floats, the biggest number being the recognized breed.
- This array is sorted and sent back to the main activity, where we are showing the first breed in a popup window, as the recognized breed.

- The recognized breed is sent as an Extra Text, to the History Activity, where the text is extracted, and added to the RecyclerView list.
- A RecyclerView has three items: The text that is extracted from the intent received from the main activity. This text is used for mapping a generic picture representing the breed recognized, which is the second element of the RecyclerView. The third is a button used for deleting items from the list, that looks like a red cross.
- For saving our Data, we use ROOM, which is an abstraction level over SQLite. The data is saved in one entity table, named “breeds”, that uses the timestamp as the primary key, and has description as another field.
- Our application also uses the MVVM architectural pater.
- The User Interface is extremely simple, there is only one floating button in each screen, that is used to move to the other screen, and the button that deletes the breed recognized from the history list.
  To ease the code surfing, here is a simple diagram of the actions between activities, including the classifier and the RecyclerView List



This diagram doesn’t include the Room model, with its entity and dao, but includes the class from the ViewModel that uses the RecyclerVIew list as an observable.

State of the Art

Characteristics	**Dog Scanner**	**NeuroDog**	**MyApp**
Store Link	Google Play	Google Play	-
Store grade	4.3	3.3	-
Nr. installs	100M+	500+	-
Nr. ratings	12k	9	-
In app ads	x	x	
Quick Recognition		x	x
Saving History	x		x
Pleasant UI	x		x
High accuracy model	x		
Opens images from galery	x	x	
Using both cameras	x		

There are just a few similar apps available right now, because the labeled data sets are not many, so it's hard to obtain a decent accuracy.

For comparison, there is the most rated app in this field, named Dog Scanner, which has a way better neural network. During our tests, Dog scanner never failed. It also has the option to run the recognize algorithm on images from the phone gallery, and has a better build User Interface. It is able to use both of the cameras(front and back). The only problem with this app, is the fact that it takes some time to run the recognition, around 10 seconds. It also has a version that you can buy.

The NeuroDog application is an open-source application from which we took the Neural Network algorithm. It has a pretty bad user interface, and it's not saving the recognized breeds, but it's really fast. Unfortunately, the accuracy of the network is low, the application is missing a lot. It has the option to run the algorithm on pictures from the phone gallery.

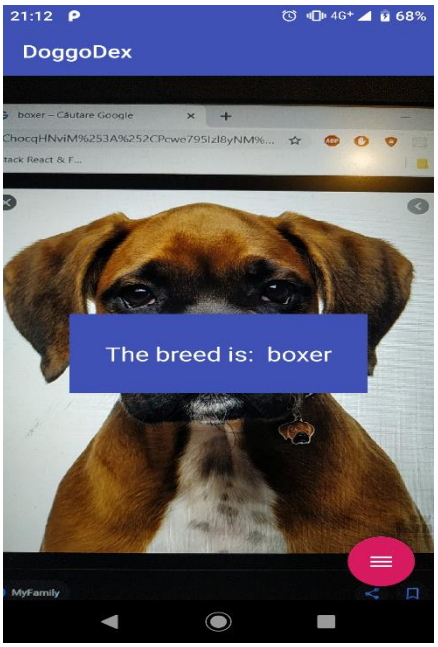
Usage

There are two screens. The first one is the one that opens a CameraView and takes the picture:

It has only one button, the one that opens the History activity.

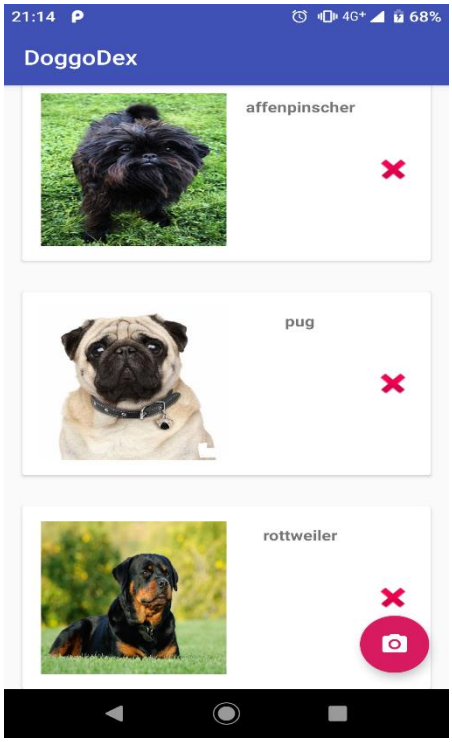
The usage of this screen is simple, the user point the camera to the dog (in this case a picture of a boxer), and taps on the screen.

After showing the breed recognized in a Popup Window, the History screen is opened, with the new breed being added into the list.



The other screen is the History one:

It has two buttons. One that opens the Main activity, so you can recognize another breed, and one that deletes breeds from the History list.



Conclusions

As a conclusion, we think that our application, and the entire field of image recognition using just the camera from our mobile phone, has a lot to grow. The only problem for now is that there are just a few labeled datasets.

Despite the fact that we have no knowledge and no experience with Kotlin and with mobile applications in general, we managed to make our app do a basic recognition on an image, which I think is a win.

There are many features that could be added, for example more breeds, or some information about each breed. Maybe the application could save the location and the dog that you recognize, and become some kind of social network, having as friends the dogs and their owners that you meet while walking your own dog. As another feature, it could use the location, as a fitness app, saving the path that you use for walking your dog, and it could show you statistics as steps, or kilometers.

The lab was awful at start, because we started with no knowledge, but it turned out to be very useful, once we started to understand a little bit. All of the subjects where actual, and the technologies were state of the art, but a slower pace would be ideal, especially because everything is new.

References

The dataset used for training the network:

[1]-https://www.kaggle.com/c/dog-breed-identification/data