A progressive web application for bird classification

Interim Project Report

TU858

BSc (Hons) in Computer Science

Robin Huumonen

Michael Collins

School of Computing

Technological University Dublin

7.3.2021

Abstract

This project report details the process of designing and developing a progressive web application (PWA) for bird species classification. The goal of this project is to provide full offline classification capabilities, once the classifier has been stored in client-side storage.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

Robin Huumonen

7.3.2021

Acknowledgements

I would like to thank Drs Michael Collins and Emma Murphy for their guidance in project management, academic writing and studies in general.

Table of Contents

1. Introduction

1.1. Overview and Background

Progressive web applications have been here for a while now, whether we have noticed them or not. Companies such as Twitter, Starbucks, Uber and Washington Post have PWAs accompanying native apps or have completely switched over. One of the major benefits of PWAs is that mobile users can add them to their home screen among native apps. The difference is that PWAs are still web applications, but with their features they can be developed to resemble native apps. With different caching strategies enabled by the use of a Service Worker, PWAs can be developed to have different levels of offline capabilities (Tamire, 2019). This feature is used to make this project a classification tool to be used in places where there might not be reliable internet connection.

According to John White's sourced blog post, bird watching, also known as birding, is one of the fastest growing hobbies in North America. In fact, Canadians spent more time birding than gardening (White, 2019). There are many available groups, clubs, books and apps for the hobbyists. Many apps provide help to locate and identify bird species. Some apps can classify birds with automatic image or sound recognition, others can help to recognize by asking features of the observed bird and narrowing down one's options (Scott, 2021).

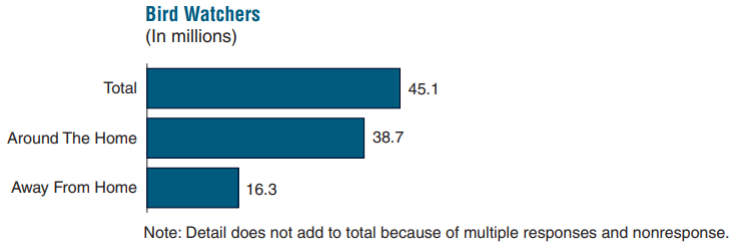


Figure 1: Bird Watching Participants in the USA (U.S. Census Bureau, 2016)

Classifications will be done with Google's open source TensorFlow (TF), a platform for building and running deep learning models. Models are trained with machine learning algorithms; they encapsulate all the information required to make predictions of the data that is run through it (Brownlee, 2019b). TF also provides a JavaScript (JS) version that can be used in a web browser or Node.js environment. This project will run a TF.js

model in a web browser to make client-side offline classification possible. On-device computation is great for data privacy and accessibility (Smilkov et al., 2019)

1.2. Project Objectives

The objective of this project is to develop and deploy a PWA that is capable of on-device classification of bird species. The aim is to train an accurate multinomial classification model with the data set selected in the research process. As for the web development of this project, the goal is to include enough PWA features such that the application will be identified as progressive by web browsers and Lighthouse audit.

1.3. Structure of the Document

In *Research* chapter the technologies selected are React, Express and Keras. MobileNetV2 convolutional neural network (CNN) architecture is introduced. In research for MobileNetV2 some prerequisite information is presented to understand the architecture. *Design* chapter presents the project management methodology used, architectures, UI design and use cases. The project will use a simple agile methodology. Service Worker's caching will be explained and TF.JS architecture introduced. UI design is done with low fidelity prototype and current screens of the application displayed. Features are shown with use cases. In *Development* current work is detailed with some code snippets and explanations. Front end is still in progress and back end yet to be commenced. The classifier is developed and its training history plotted. *References* were done with Biblatex package with Biber back end and authoryear style. *Appendix* has one appendix containing most of the Keras training script.

2. Research

2.1. Overview of Technologies

Research for developing a PWA started with choosing a front end framework. React was chosen, because starting development environment using integrated tool chain Create React App with built-in PWA template is a great starting point for developing a PWA with offline-first caching (Timothy, 2020). It initiates one of the core features of PWAs, a Service Worker, that is a script that runs in the background in a separate thread. It enables different caching strategies by intercepting request-response flow (Tamire, 2019). Also, a web app manifest file will be written. It is a file that gives information about the PWA so that it can be 'installed' on a user's device. A typical manifest file includes the app name, icon and the URL that should be opened when the icon is clicked (Beaufort, 2021).

For the back end Express framework for Node.js was chosen, because it is a minimalist framework, and the author is familiar with it. Server's role in this project is minimal. It will only serve the website's code and the TF.js model. Nothing will be uploaded to the server. The web page assets, such as JS and HTML, will be stored to a browser's cache using Cache API. The classification model will be stored with IndexedDB API, an API for NoSQL database in a browser. The image that is about to be classified remains on-device and classification is performed in-browser. The final version of the application will be deployed on Heroku, that is a cloud platform as a service. Deploying means that the application will be usable through the Internet.

After PWA development research TF.js was researched, mainly from (Smilkov et al., 2019). TF.js is an open source library for building and running machine learning algorithms in a web browser or Node.js environment. This project will run a TF.js model in a web browser to allow offline classification. TF.js has a model converter that can convert pre-trained TensorFlow or Keras models into TF.js web format. The model for the project was chosen to be developed with Keras, which is a more user-friendly and easy to use high level Python API for Tensorflow (Gadicherla, 2020).

Data set was searched from Kaggle, a community for people sharing data sets and code to perform machine learning. The selected data set is of 250 bird species, with 35

215 training images, 1250 test and validation images (5 per species). Training images' distribution of images per species is uneven, but at least 100 training images per species is assured, and the creator of the data set states that he has been able to develop an accurate classifier despite this. Another imbalance of this data set is with the sexes. 80 of the images are from male birds. Male birds typically are more brightly colored than their female counterparts, thus they may look very different. The creator warns that due to this a classifier trained with this data set may not perform as well on female birds. (Piosenka, 2021).

Finally different CNN architectures were researched. Google's MobileNetV2 was chosen because it is designed to be very small and have low latency, especially with mobile and embedded vision applications in mind. MobileNets are nearly as accurate as the well-known VGG16, and the first version is 32 times smaller and 27 times less compute intensive (A. G. Howard et al., 2017). Newer MobileNet retains the same accuracy while being faster and having less parameters, which leads to smaller size.

2.2. MobileNetV2

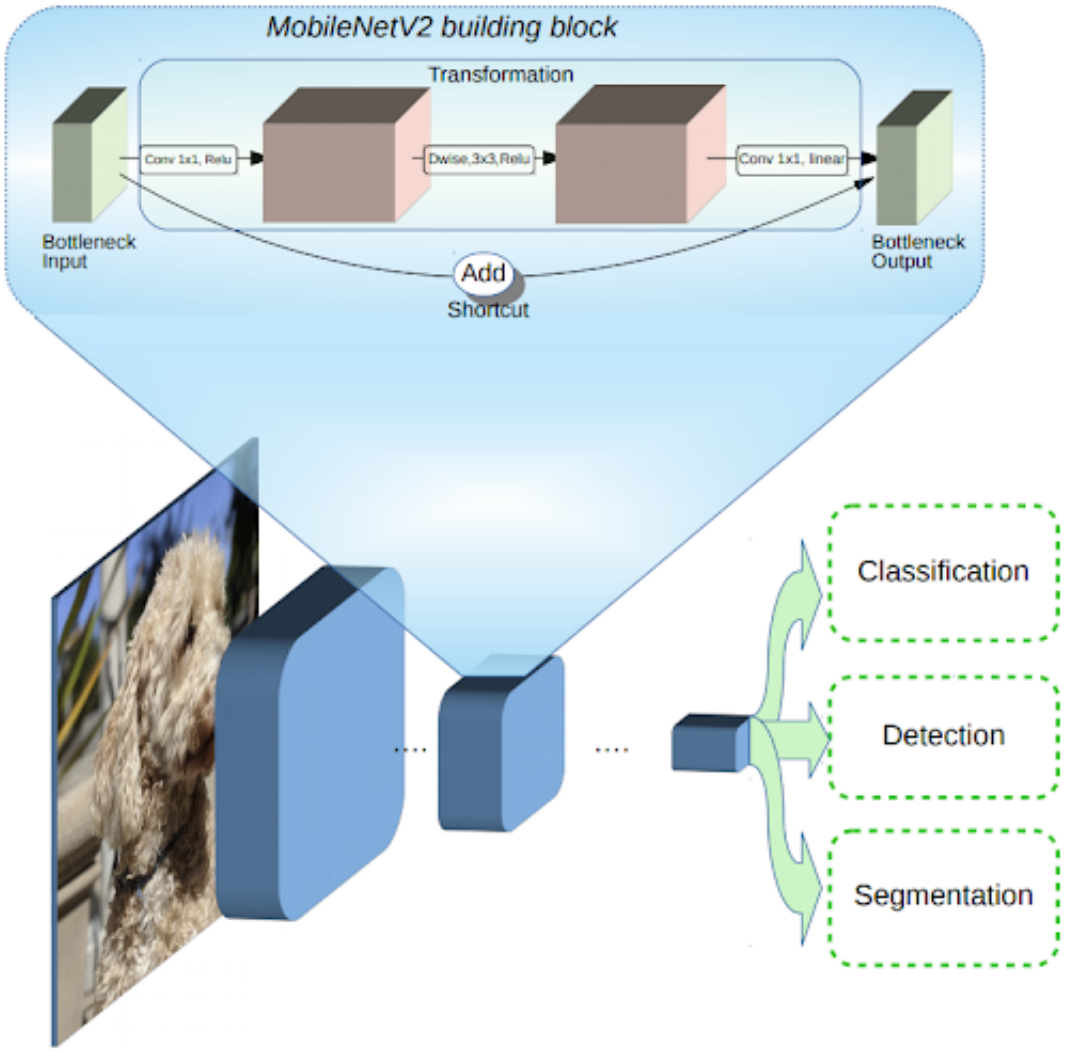
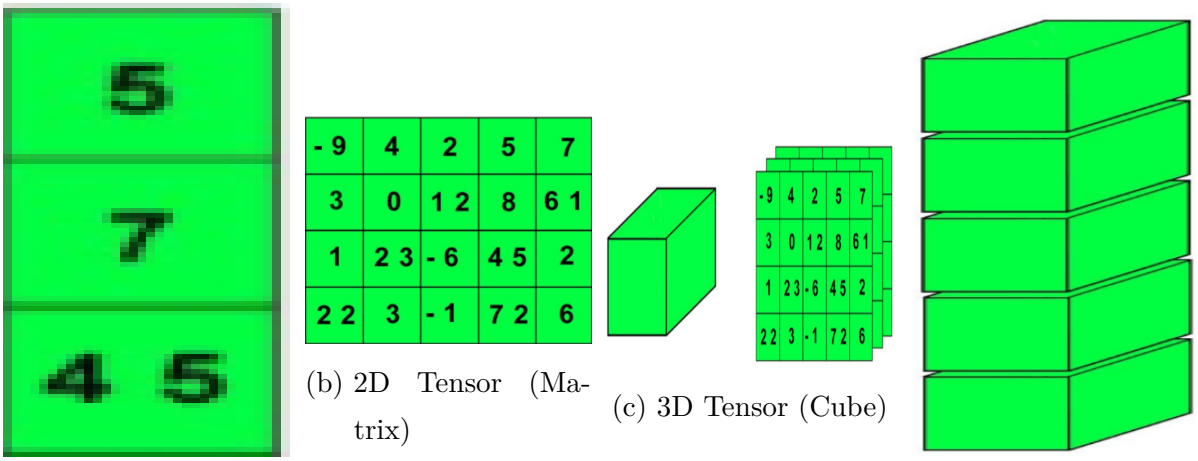


Figure 2: Overview of MobileNetV2 Architechture (M. S. A. Howard, 2018)

To get the idea of MobileNetV2 architecture (Figure 2) following concepts will be introduced: tensor, 2d convolution, depthwise separable convolution, inverted residual and linear bottleneck layer. Where the last two were introduced in MobileNetV2 as its building block.



(a) 1D Tensor (Vector)

(b) 2D Tensor (Matrix)

(c) 3D Tensor (Cube)

(d) 4D Tensor (Vector of Cubes)

Figure 3: Tensors of 1, 2, 3 and 4 Dimensions (Jeffries, 2019)

Simple way to think about tensors is that they are data structures for any kind of data. Tensors can be represented in programming with arrays of N dimensions. A color image is represented in a 3D tensor (Figure 3c). Matrices (Figure 3b) representing the image are of the height and width of the image in pixels, i.e., each element has a corresponding pixel. A pixel is the smallest unit of an image. The numerical value in an element is the color intensity of its corresponding pixel. Color image has 3 matrices, one for each color channel: red, green and blue (Bortolossi, n.d.). A set of images is represented as a 4D tensor, i.e., an array of images (Figure 3d). This is the data type fed into the CNN (Jeffries, 2019).

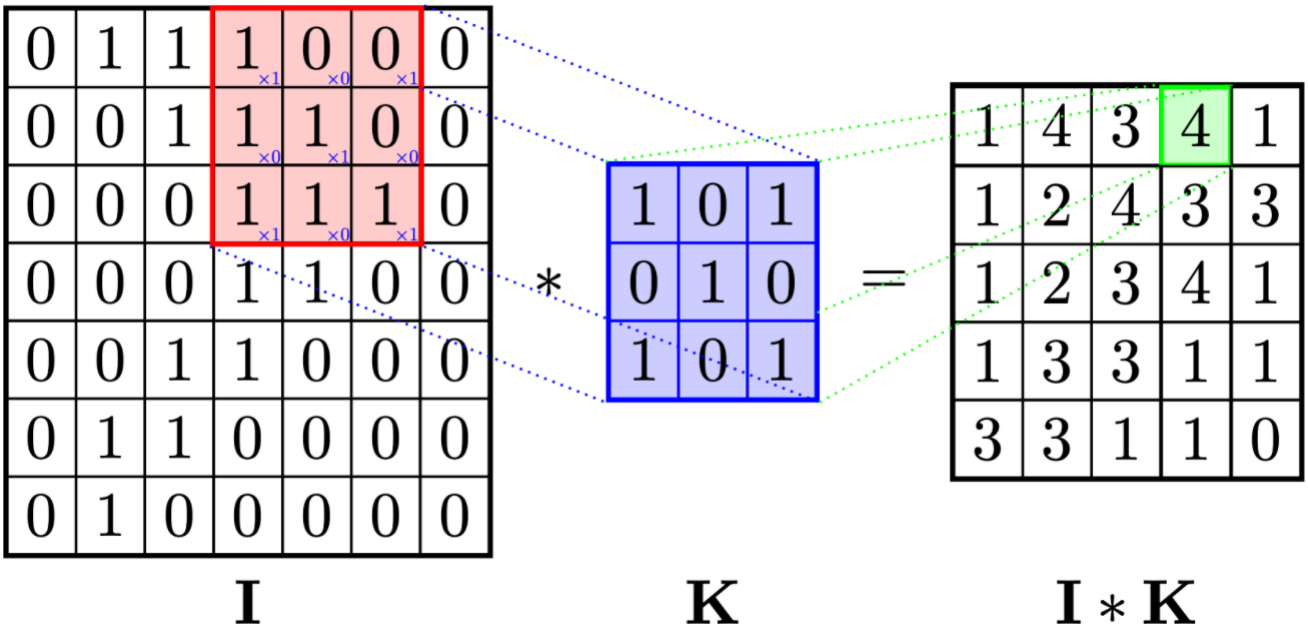
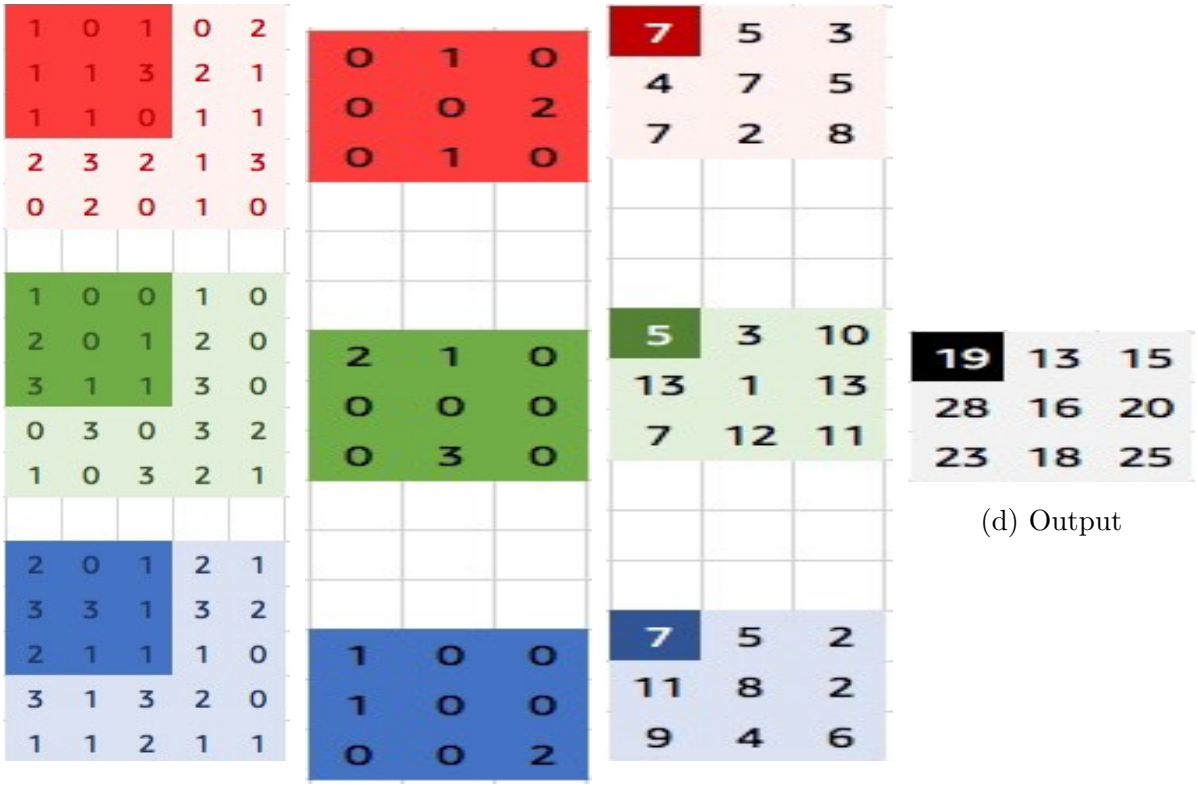


Figure 4: Matrix Convolution (*Visualizing matrix convolution* 2019)

Convolution is used in deep learning to extract features from an input image. It is element-wise multiplication and addition operation of tensors to produce an output tensor (Figure 4). The width and height of the multiplicand and the multiplier are usually different, this means that the multiplier must be moved through the multiplicand to go through all of its elements.



(a) RGB Input Channels

(b) RGB Kernels

(c) Intermediate Outputs

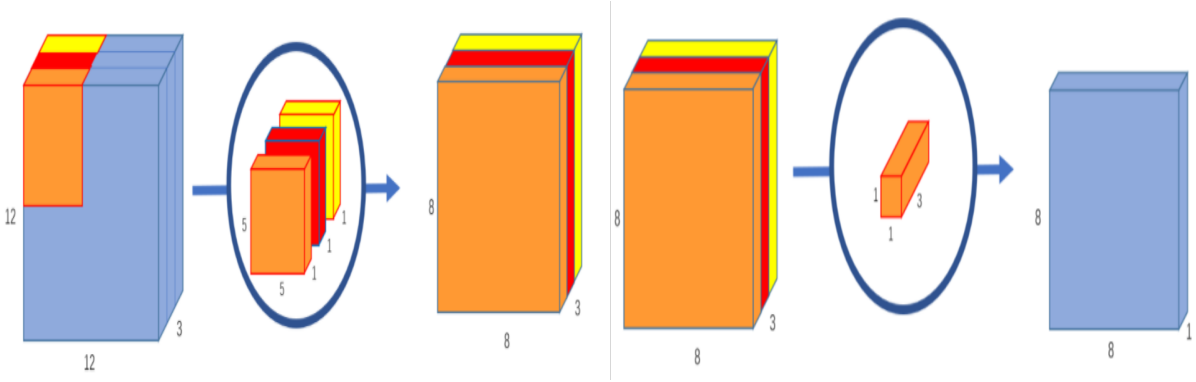
(d) Output

Figure 5: Illustration of Multichannel 2D Convolution (Lane, 2018)

In deep learning parlance, 2D and 3D tensors of weights are called kernel and filter. Weights are the parameters that the CNN is optimizing to extract enough information of the input to make predictions about it. 2D convolution means that the kernel is moved through the input in 2 directions, thus the depth (amount of matrices) of the filter and input must match (Figure 5), otherwise the kernel must be moved in 3 directions, when filter depth $<$ input depth, to go through all of the input's elements. The output of a multichannel 2D convolution operation for an RGB image is a matrix representing a greyscale image (Figure 5d). It is the sum of the intermediate outputs (Figure 5c). (Bai, 2019).

If one wants to increase the number of channels in the output image, one needs to increase the number of filters. This can become computationally heavy. Figure 5 shows the calculation of the first element of the output image using one 3-kernel filter. The MobileNetV2 initial fully convolution layer has 32 filters (Sandler et al., 2018). Mo-

bileNets started to use depthwise separable convolution to reduce computation (A. G. Howard et al., 2017). Depthwise separable convolution separates normal convolution into 2 parts: a depthwise convolution and a pointwise convolution (Wang, 2018).



(a) Depthwise Convolution (b) Pointwise Convolution

Figure 6: Depthwise Separable Convolution (Wang, 2018)

Depthwise convolution (Figure 6a) is similar to what was showed in multichannel convolution (Figure 5), but the output will have the same amount of channels as the input and filter. Convolution is done for each filter's kernel with one and only one of the image's channels. Pointwise convolution (Figure 6b) is done with the output of the depthwise. Its kernel size is 1×1, and filter depth = input depth. The output of this operation can be elongated by increasing the filters (Figure 7). In normal convolution the input image is transformed as many times as there are filters, whereas in separable convolution the image is transformed once in the depthwise step, and lengthened with the pointwise step. (Wang, 2018).

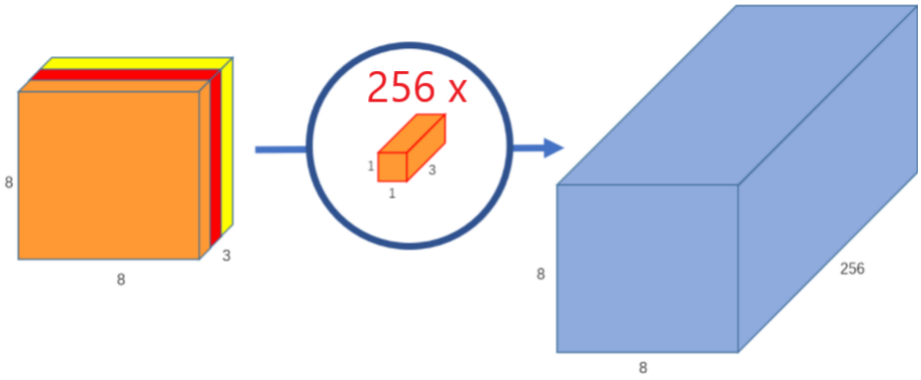


Figure 7: Pointwise Convolution with 256 Filters (Wang, 2018)

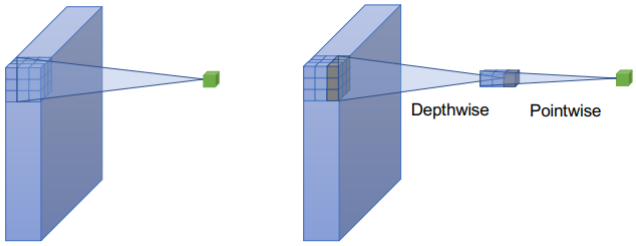


Figure 8: Normal vs. Depthwise Separable Convolution (Guo et al., 2019)

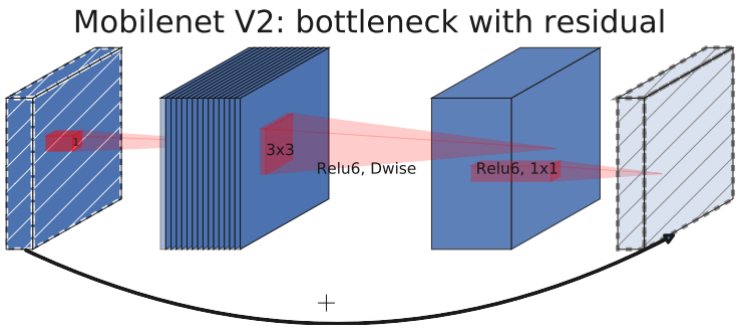


Figure 9: Inverted Residual Block (A. Howard et al., 2019)

MobileNetV2's building block (Figure 9, cf. Figure 2) follows a narrow layer ⇒ wide ⇒ narrow approach, where narrow refers to pointwise convolution and wide to depthwise. A skip connection connects the first and the last layers, the linear bottlenecks (Pröve, 2018). This allows the network to access earlier activations that were not modified in the block. Activations are the values of activation functions, which decide whether to fire a neuron.

The block uses rectified linear unit (ReLU) types as activation functions. It is simply $y = \max(0, x)$, meaning it will output the input directly if it is positive, otherwise zero (Liu, 2017). ReLU6 is defined as $y = \min(\max(0, x), 6)$, i.e., the maximum output is 6. The bottleneck layer is a layer with fewer neurons than the layer following or preceding it. These layers are used to reduce the number of feature maps, i.e., channels in the CNN, meaning the tensorial feature representations are compressed (Slater, 2017). In research for MobileNetV2 introduction paper their experimental evidence suggested that inserting linear bottleneck layers into the convolutional blocks "prevents nonlinearities from destroying too much information" (Pröve, 2018).

3. Design

3.1. Methodology

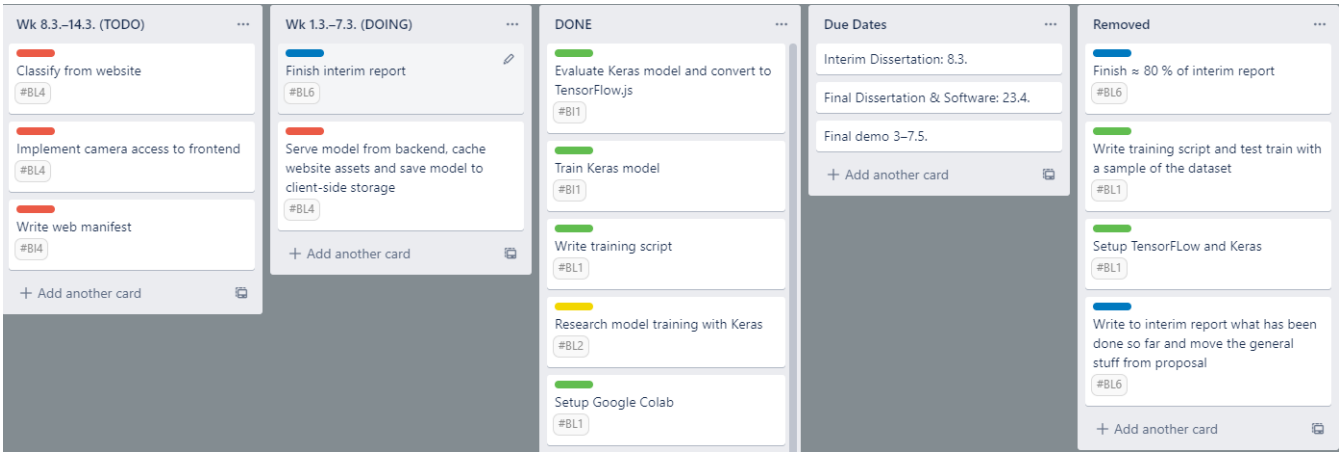


Figure 10: Modified Kanban Board

This project is being done using author's own version of Kanban methodology. Kanban is a simple agile methodology, which has a board with a three-step workflow: To Do, Doing and Done (RADIGAN, n.d.). High-level tasks are told as stories in Trello. Stories will be divided into subtasks, that can be in a To Do, Doing, Done or Removed state. Kanban board's To Do list was divided into weekly sprints during the project's development. This project will use Git for version control and GitHub for remote repositories. There will be one individual repository for the frontend, backend and classifier. These are available from: frontend, backend and classifier repositories.

3.2. Architecture

Figure 11: Web Stack

Figure 11 shows the web technologies used in this project that were introduced in the Research chapter.

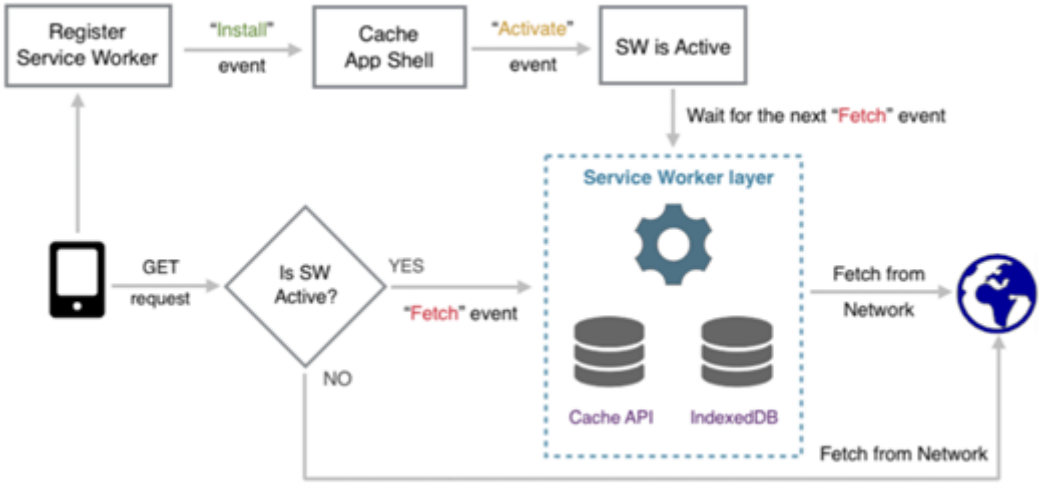


Figure 12: Service Worker's Flow for "Cache Then Network" Strategy (Idakiev, 2016)

This project will use cache then network, or offline-first, caching strategy (Figure 12). All static site assets, such as JS, HTML, CSS and image files, that are part of the bundled web application, are cached. Thus, the application loads faster on subsequent visits, and will work regardless of the network state (Timothy, 2020). A offline-first Service Worker checks whether requested files are the in cache or client-side database first before requesting them from the back end. With Cache API one can cache static assets, and with IndexedDB, one can store almost any kind of data.

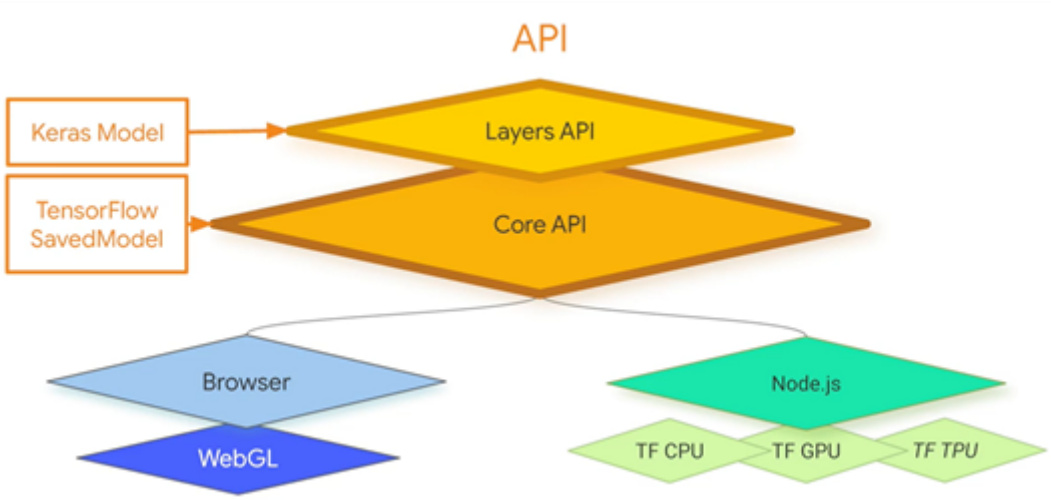


Figure 13: TF.js Architecture (Tirth1306, 2020)

In this project TF.js will be used in a browser (Figure 13). When TF.js is run in a browser, it utilizes the GPU of the device via Web Graphics Library API. TF.js offers a layers API for Keras users, that is developed to be as similar as possible to Keras (TensorFlow, n.d.).

3.3. UI

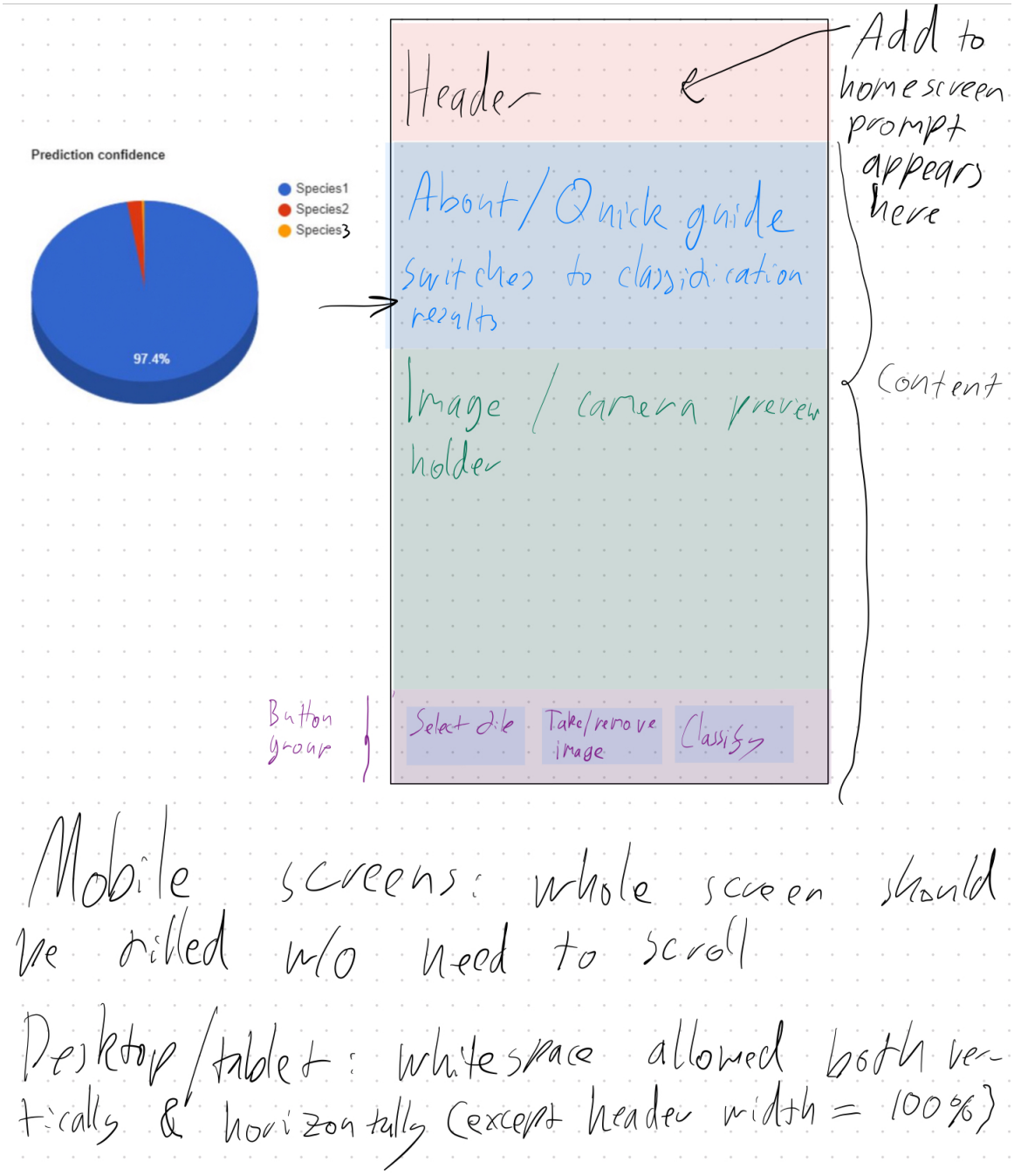
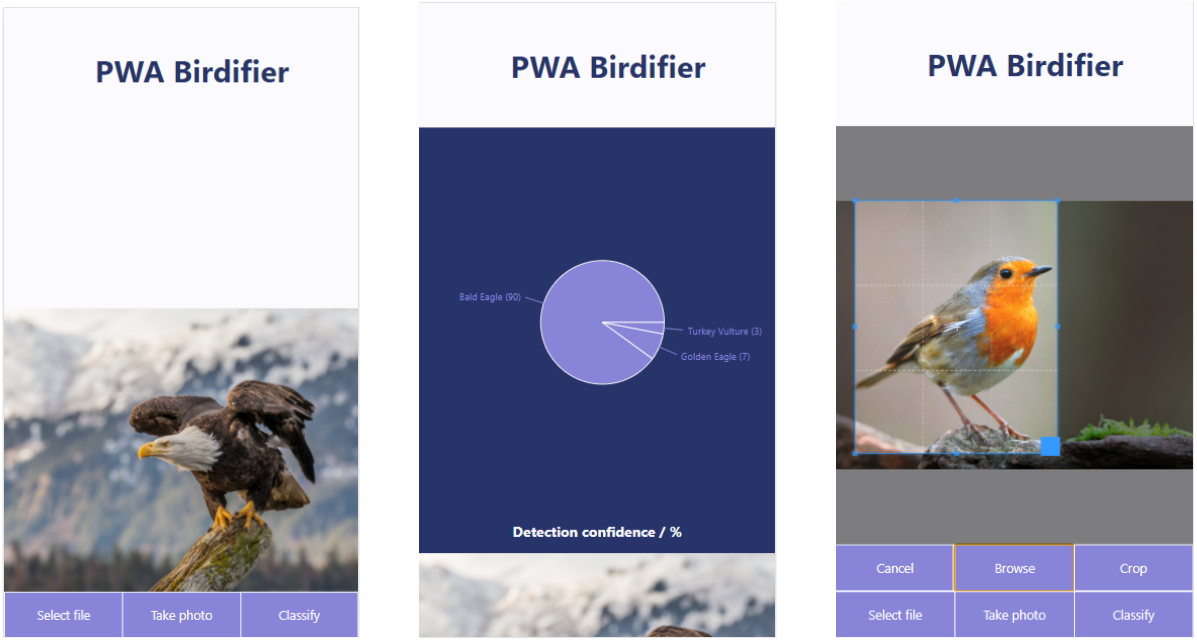


Figure 14: Low Fidelity Prototype

The UI's low fidelity prototype (Figure 14) was designed mobile-first. The React front end is a single page application (SPA), meaning the current page is updated dynamically

piece-by-piece instead of updating the whole page. About nor quick guide portions were not implemented due to the application's simplicity.



(a) Home (b) Classification Results (c) Image Cropper

Figure 15: UI Screens

Figure 15 shows the current design of the UI. Some fine-tuning will be done at the end of the project.

3.4. Features

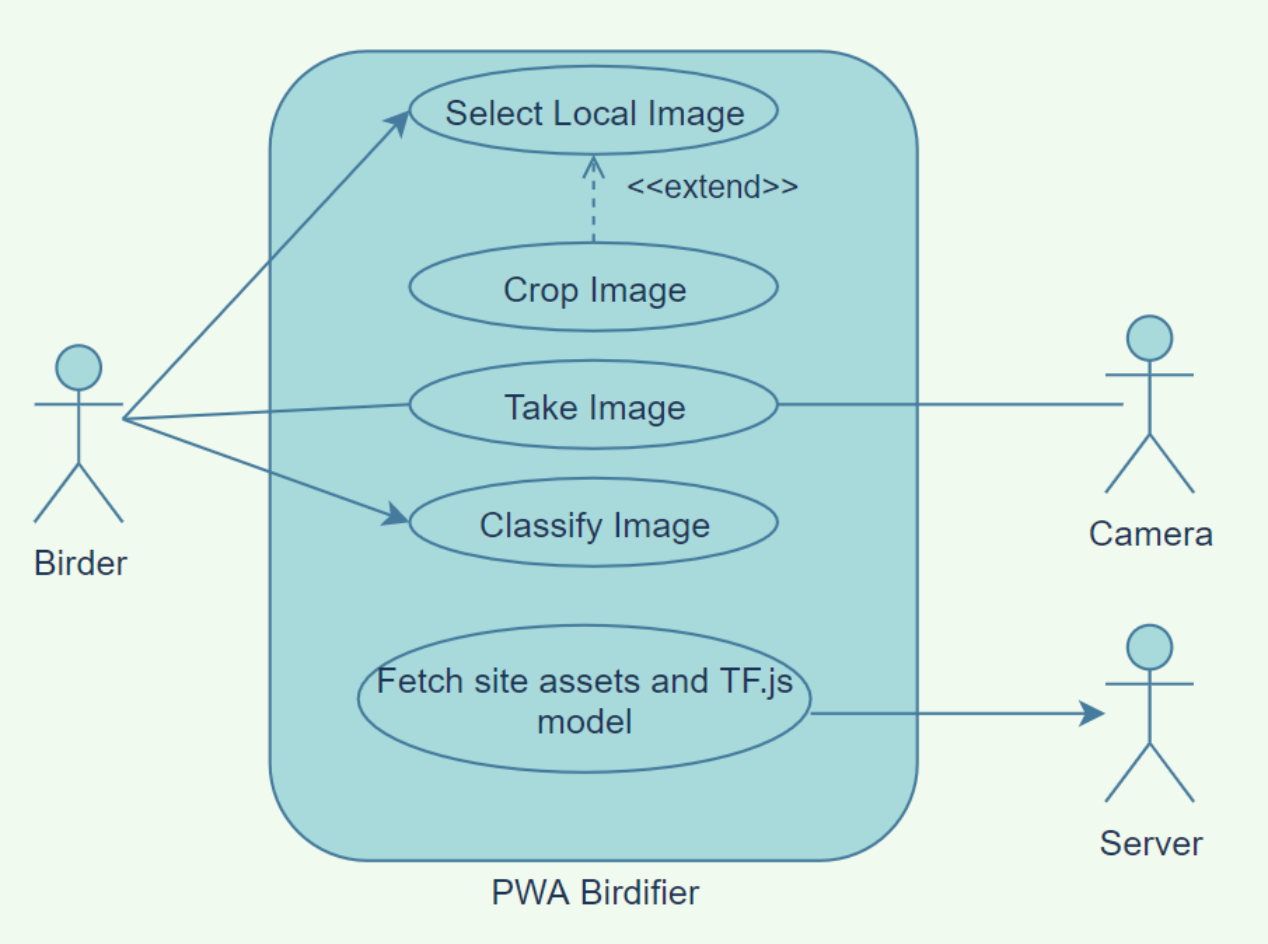


Figure 16: Use Cases

The application holds one image in cache that can be classified. The image can be the preloaded one, selected local image or just taken image. Server is thought as an external actor (Figure 16), because after the initial visit the website assets and TF.js model are stored into the system, making it an independent entity.

4. Development

4.1. Front End

```
function App() {

  useEffect(() => {
    document.title = "PWA Birdifier"
  }, []);

  return (
    <PageWrap>
    <Header/>
    <Content/>
    </PageWrap>
  );
}
```

Snippet 1: App.js

Front end development environment was booted up with *npx create-react-app myapp –template cra-template-pw* (Timothy, 2020) to get a good starting point for a Service Worker. As shown in the Design chapter the page structure is simple. React components are used to split the UI into independent pieces of code. In Snippet 1 `App()` is the root component that imports all the other components. `<PageWrap>` is just a div element with a custom style.

```
import styled from 'styled-components';
export const ContentWrap = styled.div`
  flex: 1;
  display: flex;
  flex-direction: column;
  min-height: 100%;
  width: 100%;

  div {
    flex: 1;
  }

  @media(min-width: 1024px) {
    width: 40rem;
    margin:0 auto;
  }
```

Snippet 2: ContentStyles.js

All components are styled with styled-components package as shown in Snippet 2.

```
ReactDOM.render(
  <React.StrictMode>
  <App />
  </React.StrictMode>,
  document.getElementById('root')
);
serviceWorkerRegistration.register();
```

Snippet 3: Index.js

The root component (Snippet 1) is rendered, i.e., transformed for presentation, in the index.js file (Snippet 3), which is analogous to a index.html file. The Service Worker is also registered with ServiceWorkerRegistration interface here.

```
return (
  <ContentWrap>
    {renderResults === true ?
      <Results classifications={classifications}/>
    : null}
    {selectImage === true ?
      <SelectImage setSelectImage={setSelectImage} setImg={setImg}/>
    : <div className="currentImage">
        <img src={img}></img>
      </div>}
    <div className="btn-group">
      <button onClick={() => selectImageOnClick()}>Select file</button>
      <button onClick={() => setRenderResults(false)}>Take photo</button>
      <button onClick={() => classify()}>Classify</button>
    </div>
  </ContentWrap>
);
```

Snippet 4: Content.js

Camera access nor image classification are not yet implemented in the Content component (Snippet 4). The component is rendered conditionally, meaning a different view is returned depending on the state of the component. Conditional rendering was done with ternary operators, of which syntax:

`condition ? exprIfTrue : exprIfFalse.`

Local image cropping was made with react-cropper (Kuanghuei, 2021) and pie chart with recharts (Recharts, n.d.) libraries. Code for these was adjusted from their working examples.

4.2. Classifier

The bird classifier was developed in a Google Colaboratory (Colab) environment. With Colab one can execute Jupyter notebooks on Google's cloud servers. Servers can be hardware accelerated with graphics or tensor processing units. This project's classifier was trained with TF GPU and Keras library. Most of the training script can be read from Appendix A.1.

Image data was read from directories using Keras's ImageDataGenerator. RGB intensity coefficients were rescaled from 0–250 to 0–1 to make them easier to process for the model. MobileNetV2 was used as the model, which was described in the Research chapter. The classifier's training process was started with pre-trained ImageNet weights. Starting with weights already trained to some task and using the general knowledge that the CNN has learned to a new task is called transfer learning (Brownlee, 2019a). ImageNet is a huge image dataset, thus models trained with it knows how to recognize many features. The last layer in the model has as many neurons as there are classes (250), and it uses Softmax activation, because the classifier trained for multi-class classification. Softmax, or normalized exponential function, returns the probability distribution of a list of potential outcomes (Uniqtech, 2018).

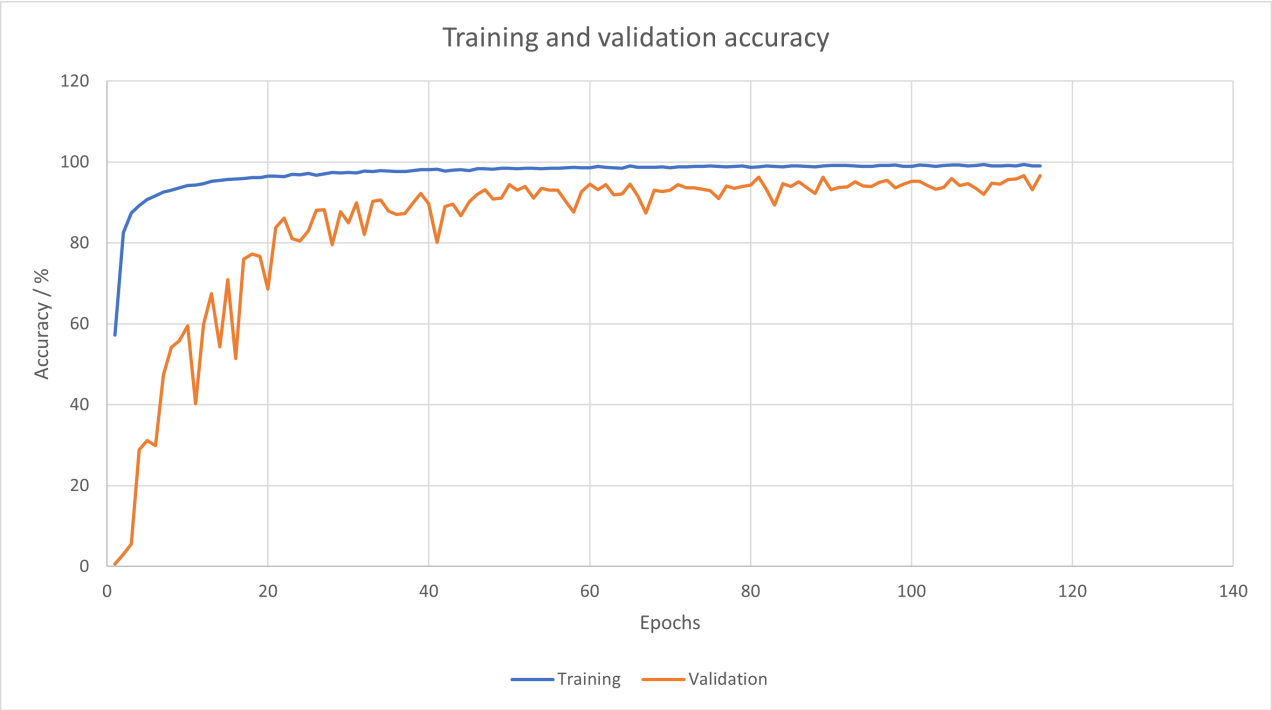


Figure 17: Accuracy

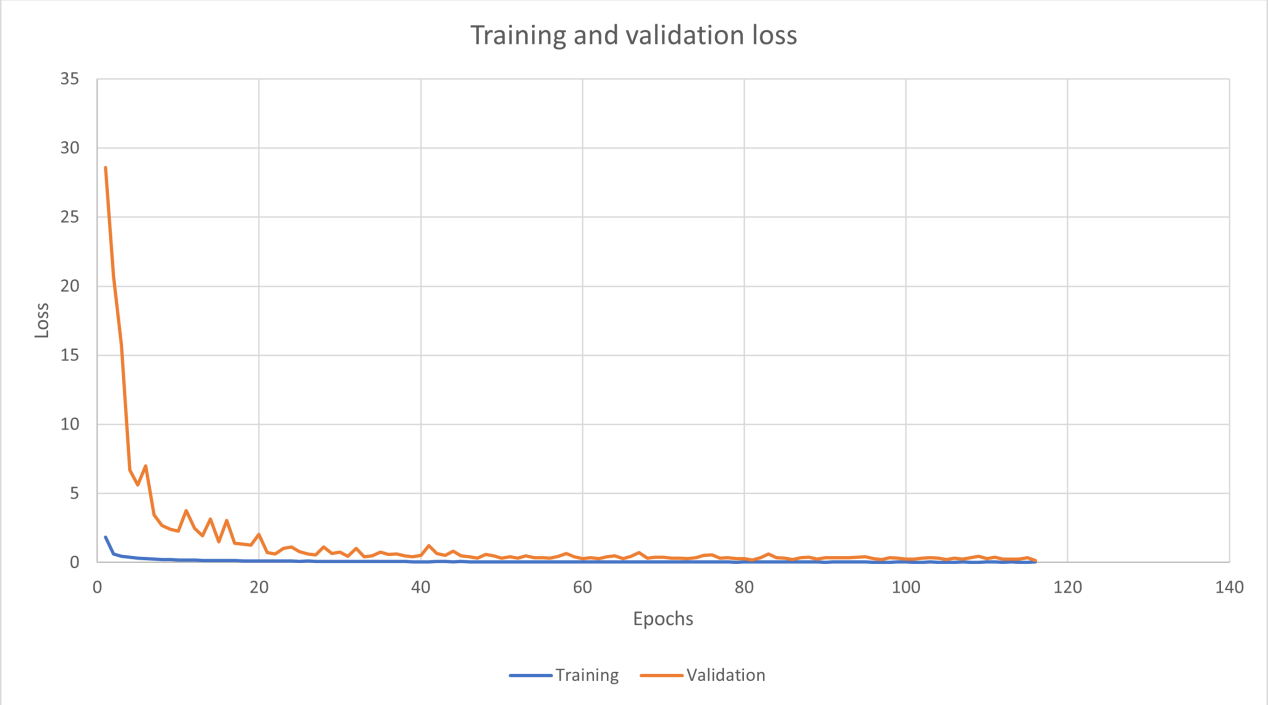


Figure 18: Loss

During training validation loss was monitored with EarlyStopping callback. Validation data is used to fine-tune the model with hyperparameters to data it has not used in training, because any good model will fit to the training data almost perfectly, thus it is overfitted to the data, meaning it will predict poorly on data it has not seen. Hyperparameters are used to control the training process, whereas weights are derived via training. Loss was monitored because it measures how bad the model's predictions are. Model fitting history was saved into a file with CSVLogger callback and plotted with Excel (Figures 17 and 18). (Claesen and Moor, 2015).

Patience in the EarlyStopping callback was set to 40 epochs, which is large for a training process like this, but there was no hurry to train the model in the cloud. One epoch means that the entire data set goes through the model once. The training was stopped by the EarlyStopping callback at the 156th epoch and best weights were restored from the 116th epoch. The model can overfit to the validation data also, although it is rare. This is why the final evaluation was done with data the model had never seen in the training nor validation process with `model.evaluate()`. This method evaluated the accuracy to 97.5 % and the loss to 0.1386. For comparison, the validation accuracy was 96.6 % and the loss 0.153 at the 116th epoch.

4.3. Future Work

The project needs a back end to send the website assets and the model to a client. Assets need to be cached and the model saved to IndexedDB. TF.js code will be written in the front end to perform classifications with the model. Also, camera access and web manifest file will be done. Other work will be conducted in the validation phase, that will be detailed in its own chapter in the final report.

References

Bai, Kunlun (Feb. 11, 2019). *A Comprehensive Introduction to Different Types of Convolutions in Deep Learning*. URL: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215 (visited on 03/06/2021).

Beaufort, Pete LePage; François (Feb. 1, 2021). *Add a web app manifest*. URL: https://web.dev/add-manifest/ (visited on 01/31/2021).

Bortolossi, Dirce Uesu Pesco; Humberto Jos´e (n.d.). *Matrices and Digital Images*. Tech. rep. Institute of Mathematics and Statistics Fluminense Federal University. URL: http://dmuw.zum.de/images/6/6d/Matrix-en.pdf (visited on 03/06/2021).

Brownlee, Jason (Sept. 16, 2019a). *A Gentle Introduction to Transfer Learning for Deep Learning*. URL: https://machinelearningmastery.com/transfer-learning-for-deep-learning/ (visited on 03/07/2021).

– (Aug. 19, 2019b). *Difference Between Algorithm and Model in Machine Learning*. URL: https://machinelearningmastery.com/difference-between-algorithm-and-model-in-machine-learning/ (visited on 02/19/2021).

Claesen, Marc and Bart De Moor (2015). "Hyperparameter Search in Machine Learning". In: *CoRR* abs/1502.02127. arXiv: 1502.02127. URL: http://arxiv.org/abs/1502.02127.

Gadicherla, Sameer (Jan. 23, 2020). *Tensorflow Vs. Keras: Comparison by Building a Model for Image Classification*. URL: https://hackernoon.com/tensorflow-vs-keras-comparison-by-building-a-model-for-image-classification-f007f336c519 (visited on 01/31/2021).

Guo, Yunhui et al. (2019). "Depthwise Convolution is All You Need for Learning Multiple Visual Domains". In: *CoRR* abs/1902.00927. arXiv: 1902.00927. URL: http://arxiv.org/abs/1902.00927.

Howard, Andrew et al. (2019). "Searching for MobileNetV3". In: *CoRR* abs/1905.02244. arXiv: 1905.02244. URL: http://arxiv.org/abs/1905.02244.

Howard, Andrew G. et al. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. arXiv: 1704.04861 [cs.CV].

Howard, Mark Sandler; Andrew (Apr. 3, 2018). *MobileNetV2: The Next Generation of On-Device Computer Vision Networks*. URL: https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html (visited on 03/01/2021).

Idakiev, Ilia (Oct. 24, 2016). *Angular Offline Progressive Web Apps With NodeJS*. URL: https://www.slideshare.net/IliaIdakiev/angular-offline-progressive-web-apps-with-nodejs (visited on 01/31/2021).

Jeffries, Daniel (Sept. 27, 2019). *Learning AI if You Suck at Math — P4 — Tensors Illustrated (with Cats!)* URL: https://hackernoon.com/learning-ai-if-you-suck-at-math-p4-tensors-illustrated-with-cats-27f0002c9b32 (visited on 03/06/2021).

Kuanghuei, Fong (Mar. 4, 2021). *react-cropper*. URL: https://github.com/react-cropper/react-cropper (visited on 03/07/2021).

Lane, Thom (Oct. 18, 2018). *Multi-Channel Convolutions explained with... MS Excel!* URL: https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108 (visited on 03/06/2021).

Liu, Danqing (Nov. 30, 2017). *A Practical Guide to ReLU*. URL: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7 (visited on 03/06/2021).

Piosenka, Gerald (2021). *https://www.kaggle.com/gpiosenka/100-bird-species*. (Visited on 01/31/2021).

Pröve, Paul-Louis (Apr. 11, 2018). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. URL: https://towardsdatascience.com/mobilenetv2-inverted-residuals-and-linear-bottlenecks-8a4362f4ffd5 (visited on 03/06/2021).

RADIGAN, DAN (n.d.). *How the kanban methodology applies to software development*. URL: https://www.atlassian.com/agile/kanban (visited on 01/31/2021).

Recharts (n.d.). *A composable charting library built on React components*. URL: https://recharts.org/en-US/ (visited on 03/07/2021).

Sandler, Mark et al. (2018). "Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation". In: *CoRR* abs/1801.04381. arXiv: 1801.04381. URL: http://arxiv.org/abs/1801.04381.

Scott (2021). *The 13 Best Bird Watching Apps (2021)*. URL: https://birdwatchinghq.com/birdingapps/#recognition (visited on 02/27/2021).

Slater, Neil (Dec. 31, 2017). *What is the concept of Tensorflow Bottlenecks?* URL: https://ai.stackexchange.com/a/4887 (visited on 03/06/2021).

Smilkov, Daniel et al. (2019). "TensorFlow.js: Machine Learning for the Web and Beyond". In: Palo Alto, CA, USA. URL: https://arxiv.org/abs/1901.05350.

Tamire, Workneh (Dec. 16, 2019). "Evaluation of Progressive Web Application to develop an Offline-First Task Management App". en. G2 Pro gradu, diplomityö, p. 74. URL: http://urn.fi/URN:NBN:fi:aalto-201912226568.

TensorFlow (n.d.). *TensorFlow.js layers API for Keras users*. URL: https://www.tensorflow.org/js/guide/layers_for_keras_users (visited on 01/31/2021).

Timothy (Sept. 9, 2020). *Making a Progressive Web App*. URL: https://create-react-app.dev/docs/making-a-progressive-web-app/ (visited on 03/01/2021).

Tirth1306 (Sept. 3, 2020). *Browser-based Models with TensorFlow.js*. URL: https://medium.com/analytics-vidhya/browser-based-models-with-tensorflow-js-afcd10be3615.

U.S. Census Bureau (2016). *National Survey of Fishing, Hunting, and Wildlife-Associated Recreation*. Research rep. URL: https://www.census.gov/programs-surveys/fhwar.html (visited on 02/27/2021).

Uniqtech (Jan. 30, 2018). *Understand the Softmax Function in Minutes*. URL: https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d (visited on 03/07/2021).

Visualizing matrix convolution (Dec. 28, 2019). URL: https://tex.stackexchange.com/a/522122 (visited on 03/06/2021).

Wang, Chi-Feng (Aug. 14, 2018). *A Basic Introduction to Separable Convolutions*. URL: https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728 (visited on 03/06/2021).

White, John (Aug. 15, 2019). *How popular is birdwatching?* URL: https://chirpbirding.com/blog/81/how-popular-is-birdwatching/ (visited on 02/27/2021).

A. Appendix

A.1. Meat of the Notebook Used to Train the Model

[5]:
```
rescaled_ImageDataGenerator = ImageDataGenerator(rescale=1.0/255) #rescale=1./
 ↪255 scales RGB coefficients from 0-255 to 0-1
```

[6]:
```
pixel_height = pixel_width = 224
batch_size = 64
classes_amount = 250
color_channels = 3
training_samples = 35215
validation_samples = 1250
```

Data downloaded with Kaggle API from Gerald Piosenka

[7]:
```
train_generator = rescaled_ImageDataGenerator.flow_from_directory(
    base_dir + train,
    target_size=(pixel_height,pixel_width),
    batch_size=batch_size,
    class_mode='categorical'
)
validation_generator = rescaled_ImageDataGenerator.flow_from_directory(
    base_dir + valid,
    target_size=(pixel_height,pixel_width),
    batch_size=batch_size,
    class_mode='categorical'
)
test_generator = rescaled_ImageDataGenerator.flow_from_directory(
    base_dir + test,
    target_size=(pixel_height,pixel_width),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)
```

```
Found 35215 images belonging to 250 classes.
Found 1250 images belonging to 250 classes.
Found 1250 images belonging to 250 classes.
```

[8]:
```
#Creates MobileNetV2 model with preloaded weights. Src: https://medium.com/
 ↪hackernoon/tf-serving-keras-mobilenetv2-632b8d92983c
def create_model():
  input_tensor = Input(shape=(pixel_height, pixel_width, 3))

  base_model = tf.keras.applications.MobileNetV2(
  include_top=False,
  weights='imagenet',
  input_tensor=input_tensor,
  input_shape=(pixel_height, pixel_width, 3),
  pooling='avg'
)

  for layer in base_model.layers:
```

```
    layer.trainable = True  # trainable has to be false in order to freeze␣
 ↪the layers

  op = Dense(256, activation='relu')(base_model.output)
  op = Dropout(.25)(op)

  output_tensor = Dense(classes_amount, activation='softmax')(op)

  model = Model(inputs=input_tensor, outputs=output_tensor)

  return model
```

[10]:
```
def evaluate_model(model):
 model.evaluate(test_generator, steps=len(test_generator))

  print("Test loss: {0:.4f}".format(test_loss))
  print("Test acc: {0:.4f} % ".format(test_acc * 100.0))
```

[12]:
```
model = create_model()
model.compile(optimizer = 'adam',loss = 'categorical_crossentropy', metrics =␣
 ↪['accuracy'])
print(model.summary())
```

```
=================================================================================
(Architechture summary removed)
Total params: 2,650,170
Trainable params: 2,616,058
Non-trainable params: 34,112
=================================================================================
```

[13]:
```
#Save model at some interval during the trainig process
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1,␣
 ↪save_best_only=True, mode='min')

#Use early stopping callback to avoid over fitting
stop_early = EarlyStopping(
    monitor='val_loss',
    min_delta=1e-3,
    patience=40,
    verbose=1,
    mode='auto',
    restore_best_weights = True
)

#CSVLogger logs epoch, acc, loss, val_acc, val_loss
logs = CSVLogger('/content/drive/MyDrive/Colab Notebooks/Logs/logs-v2.csv',␣
 ↪separator=',', append=False)

callbacks = [checkpoint, stop_early, logs]
```

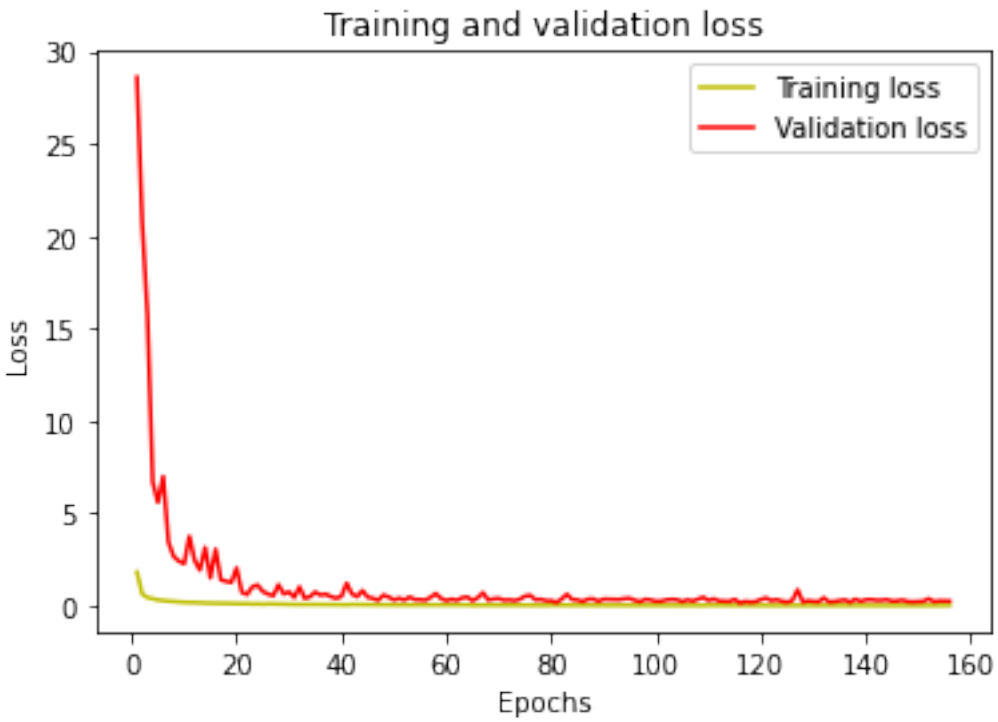
[ ]:
```
history = model.fit(
    train_generator,
```

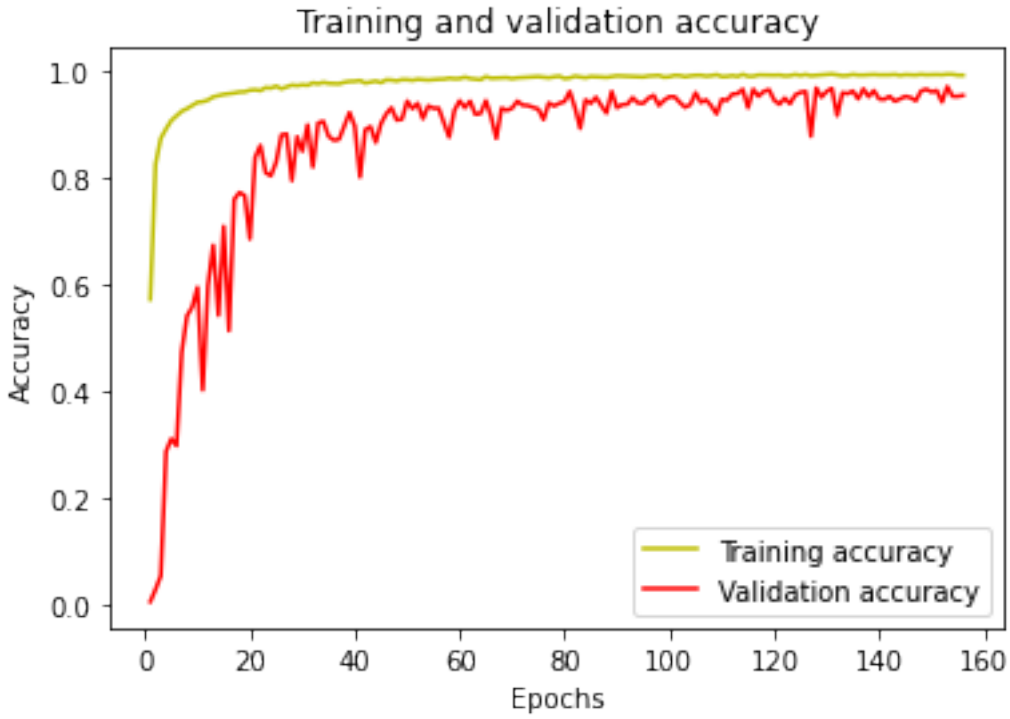
```
    steps_per_epoch = training_samples // batch_size,
    validation_data = validation_generator,
    validation_steps = validation_samples // batch_size,
    epochs = 280,
    verbose=0,
    callbacks=callbacks
)

model.save(model_name)
```

[15]:
```
evaluate_model(model)
```

```
20/20 [==============================] - 4s 182ms/step - loss: 0.1386 -
accuracy: 0.9752
Test loss: 0.1386
Test acc: 97.5200 %
```

[16]:
```
plot_model(history)
```




Training and validation accuracy
Accuracy
Epochs
Training accuracy
Validation accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120
140
160