Mansoura University

Faculty of Engineering

Communications Department

Intelligent Vision Assistant

Graduation Project

Submitted in partial fulfilment of the requirements for the degree of Bachelor of Science in Electronics and Communication Engineering

By

Asmaa Abdelhameed Mosa

Mahmoud Mohamed Ghonem

Mohamed Hany Elgharbawy

Shrook Nagy Hamed Wafa

Youssef Saad Abdulmajid

Under Supervision of

Assoc. Prof

Doaa Adel Altantawy

Electronics and Communication Engineering

Dept. Faculty of Engineering,

Mansoura University

2020

Acknowledgement

قُلْ بِفَضْلِ اللَّهِ وَبِرَحْمَتِهِ فَبِذَلِكَ فَلْيَفْرَحُوا هُوَ خَيْرٌ مِمَّا يَجْمَعُونَ

First of all, we would like to express our gratitude to ALLAH Almighty for giving us ideas and strengths to make our dreams true and accomplish this project. We wish for this project to help visually-impaired people in our Arab countries and all Arabic speakers around the world.

We have been indebted in the preparation of this project to our supervisor,

Dr. Doaa Adel,

whose patience and kindness, as well as her academic experience has been invaluable to us. We would like to express our deep and sincere gratitude to her.

We would like to thank our family, for their encouragement, patience and assistance over the years. We are forever indebted to them for always kept us in their prayers. They have been a constant source of support, emotional and moral, during our undergraduate years, and this project would certainly not have existed without them.

Table of Contents:

0. Introduction

According to W.H.O, there are 285 million visually-impaired people in the world, 12.6% of them are from the Middle East, and there are 466 million hearing-impaired people in the world, 4% of them also from the Middle East. But unfortunately, their share of technology, especially Artificial Intelligence, is very poor, all the attention at this field is focused on English speakers.

We aim by this project to help Arabic speakers by developing Arabic computer vision and Arabic NLP Algorithms which can be employed in many aspects and help many people, especially the people with disabilities.

We employed our technology in this project in helping visually-impaired people by giving them a tool that can compensate them a little for their sight.

Currency Recognition

This will help the visually-impaired people in recognizing the banknotes. We have already implanted a model that can recognize Egyptian Currency and we aim to comprise the rest of Arabic currencies in the future.

Arabic image caption

Which gives an Arabic description to the captured images.

Arabic Visual Question Answering

It makes the users able to interact with the captured images by asking a simple question about anything in the image they wonder about.

Emotion Recognition

This model will help users recognize people's feelings by detecting the faces in the captured images and analyzing facial expressions.

Mobile application

We have integrated these models in a mobile application to make it more practical and easier to use.

Considering that our models can be employed in many other applications to help different segments of society. We focused here on the Computer Vision Models to help the visually-impaired, but we can, with some additions and modifications, use them in the education of children or in helping the hearing-impaired.

1. Currency Recognition

1.1 Abstract

Money recognition is one of the most important problems facing visually-impaired people especially for paper currency. In our application we present a simple currency recognition system applied on Egyptian banknotes. Our proposed system is based on computer vision models. The experimental results demonstrate that the proposed method can recognize Egyptian paper money with high accuracy that reaches 90% and is processed in a short amount of time.

1.2 Introduction

One of the main problems faced by visually-impaired people is the impossibility of recognizing the paper currencies due to similarity of the texture and size of the paper between the various classes. The role of technology therefore lies in developing a solution to this problem in a way that makes blind people feel comfort and reliance in financial transactions.

There are two trends in research on Money Recognition; camera based and scanner based. Systems built on scanners assume all-paper capture. Such systems are suitable for Money Counters Machinery. Although, camera-based systems assume

the paper is captured through a camera capable of capturing part of the paper. Most of the related literature work deals with the scanner-based form. For visual disability usage, it is designed to encourage users to grab every part of the document on their smartphone and let the device recognize it and say the value of the currency.

Throughout this job, camera-based Egyptian paper currency is trained to be recognized using a robust convolutional neural network model, making the processing time very short with reasonable accuracy. The proposed systems have the ability to treat documents captured in part and under different lighting or locating conditions.

1.3 Related Work

Related literature work will be investigated in this section. As we mentioned earlier, most of the work consider the entire money paper (scanned paper), that is not the case with our interest. So, camera-based research is only mentioned in this section and specifically we will talk about the applications that recognize the Egyptian money.

Floos Mubser is the first Egyptian Android device to accept the Egyptian currency. Many Egyptian notes can be identified efficiently under high lighting. This has a range of functions, such as the use of camera flash lighting to provide currency identification under bad lighting. It could act with low processing power tools by collecting frames of varying quality. It can also adjust the camera contrast.

However, it lacks flexibility, since all of these features are manuals that add a heavy burden on the blind or the visually impaired to adjust and thus add complexity to the process.

Floosy is another Egyptian program. It's a smartphone Android program that has identical Floos Mubser algorithms. Uses the color features of the picture rather than the corresponding template to define the currency. For each Egyptian note, it extracts one of the color features stored in the database, automatically and periodically, frames can be captured using a device camera. By extracting a feature from each note, it compares it to its pre-calculated features and, based on similarity measures, it can find the nearest note it can be. But since it uses color technology, lighting is a key problem. It behaves differently under different lighting conditions and fails or incorrectly recognizes notes. This application has a number of features, such as the use of flash light and camera focus adjustment. These apps can be controlled manually and are versatile.

In [1] a simple currency recognition system applied on Egyptian banknote is proposed. This system is based on simple image processing utilities. The basic techniques utilized in this system include image foreground segmentation, histogram enhancement, region of interest (ROI) extraction and finally, template matching based on the cross-correlation between the captured image and their data set. The experimental results demonstrate that the proposed method can recognize Egyptian paper money with acceptable accuracy.

In this project, our proposed method considers that every part of the money paper should be captured in a particular direction. Simple CNN model is used to recognize the value of the currency after forwarding the target currency picture through our CNN model.

1.4 Methodology

In this section, we propose a general framework for identifying paper banknotes. There are 5 main stages in the system, as presented in Figure 1.

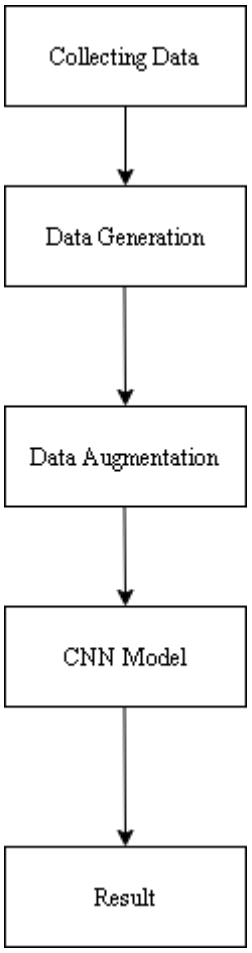


Figure (1)

1.4.1 Collecting Data

Any Computer vision model is based essentially on The Architecture (CNN model) and the dataset. We will describe our architecture in the next section with allocating this section for anything related to the data.

We need a lot of data to be able to build an accurate computer vision application, which reaches sometimes hundreds of thousands or millions. This was one of the major problems because there is no available dataset for Egyptian currency and we need, at minimum, 100,000 images to reach a satisfying accuracy.

So, we decided to build our dataset, but instead of capturing the images directly, we wrote a code using python to generate the required data.

1.4.2 Data Generation

a) Training dataset

The main idea behind generating data using code is based on the computer vision's concept for detecting and recognizing the objects at the images, that happens by recognizing the pattern in the dataset through learning process, and then the values of the model parameters is modified through forward and back propagation.

So, we need a large dataset which carries the required pattern to build the model, these images are captured, mostly, with the help of humans, which is impossible in our case because we don't have the capabilities to capture 100,000 different images.

We considered the front and the back of the same currency as a different class because they are quite different at the Egyptian currency, so they have a quite different pattern. In order to that, we have now 12 classes (5, 10, 20, 50, 100 and 200. Front and back for each). For each class, we need about 10,000 images for the training dataset and about 500 images for the validation dataset.

So, we decided to create a dataset that holds the Egyptian currency's pattern using python programming, our approach goes as follows:

1. Building a small dataset containing 18 images for each class.
2. Using COCO dataset images, and combine each currency image with a group of COCO images (about 555 images).
3. We considered all the variants through this process, first, we applied some image processing filters on the currency image before mixing it with each image from COCO dataset to make some changes at the spatial domain and intensity domain.

4. Each currency images are rotated randomly before combining it with each COCO image, and even the location of combining the two images is determined randomly.

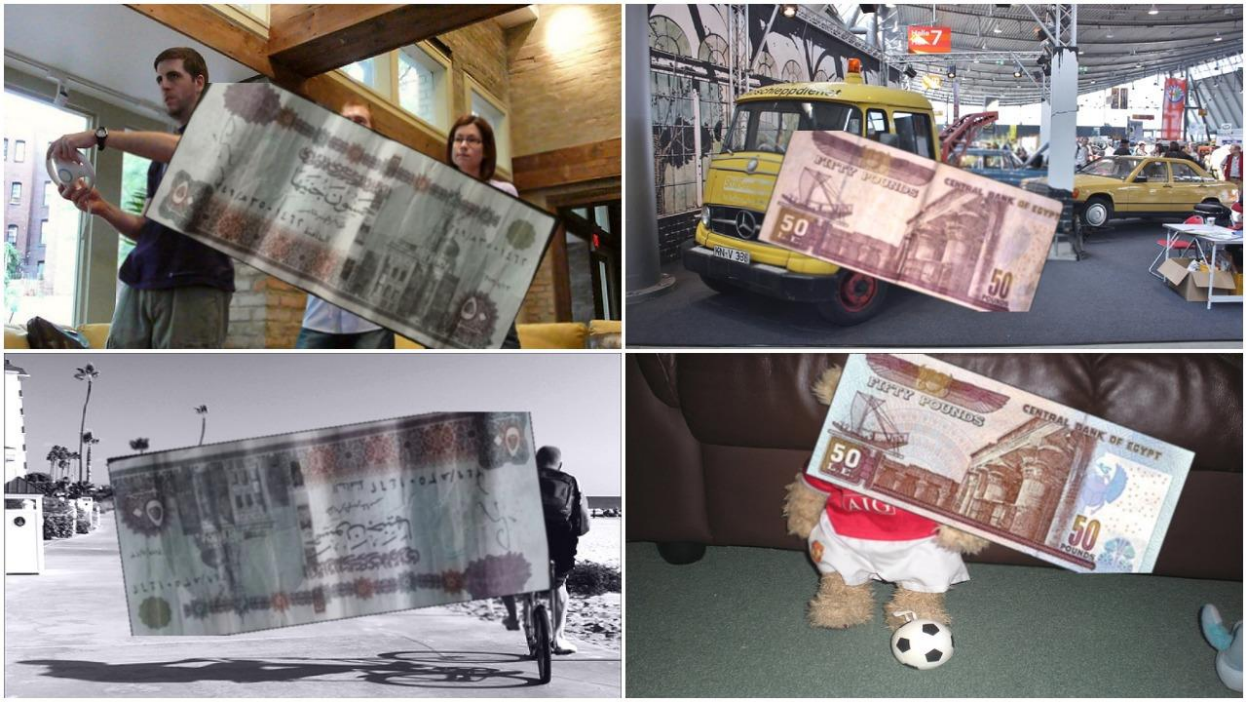
We iterate our small dataset (18 images * 12 class) over 120,000 images from COCO dataset to get a currency dataset that contain 120,000 images which is suitable enough - with using appropriate architecture with fine tuning the hyperparameters - to reach an accurate model.

These are samples of images from our dataset:

Five pounds (Front / Back)

Ten pounds (Front / Back)

Twenty pounds (Front / Back)



Fifty pounds (Front / Back)

Hundred pounds (Front / Back)

Two Hundred pounds (Front / Back)

Obviously, these images aren't realistic. the currency images have COCO images as a background, as a result we can see the currency in front of an animal, car, or even cover a person's face.

But in practice, the model considers all of these objects as a background, these weird backgrounds aren't much different from the practical one as someone who is holding the currency or the table on which it is put.

The model observes only the common patterns between the images which is the shape of the currency, with considering all object behind it as a background. Even our dataset isn't realistic for the human eye, but the model can use it through the process of learning to recognize the currency pattern with the captured images to predict the currency.

b) Validation Dataset:

Although our dataset is very appropriate in the process of learning, but we can't rely on it in evaluating the model accuracy. We need some real images to make sure that our model has reached a satisfying accuracy.

For that, we captured 5600 images (about 460 images for each class) with the help of our friends, and used it as a validation dataset.

These are samples of images from our dataset in figure (2).

Figure (2)

1.4.3 Model

Now the training dataset is available and it is time to train the model. The proposed model is mainly based on *transfer learning* using pretrained state-of-the-art models for image classification to classify custom data which in this case, is the currency dataset.

Extracting features from images by a convolutional neural network model takes a lot of time and requires expensive resources. Thus, many deep learning models resort to pre-trained image features model. We tried a number of pre-trained models such as; vgg16, vgg19, resnet101, resnet50, inceptionV3 and Xception model.

For an image of size 299x299, Inception produces a 64x2048 feature matrix and Xception produces a 100x2048 feature matrix, while VGG19 for an image of size 224x224 produces a 49x512 feature matrix, we get the best accuracy with Xception as a base model with 169x2048 feature matrix and input shape of (400,400,3), our model summary is shown in figure (3).

Layer (type)	Output Shape	Param #
xception (Model)	(None, 13, 13, 2048)	20861480
layer_normalization_1 (Layer	(None, 13, 13, 2048)	26
flatten_1 (Flatten)	(None, 346112)	0
dropout (Dropout)	(None, 346112)	0
dense_3 (Dense)	(None, 64)	22151232
dense_4 (Dense)	(None, 32)	2080
dense_5 (Dense)	(None, 12)	396

Total params: 43,015,214
Trainable params: 28,404,126
Non-trainable params: 14,611,088

Figure (3)

We changed the standard input image shape of the Xception model from (299,299,3) to (400,400,3) As it is observed that increasing the size improves the accuracy of the model, this happens as some details of the image are not clear with small size.

We used a pretrained Xception model, that architecture is illustrated in figure (4), without the classifier part with the last 17 trainable layers and the rest of layers are froze to extract feature vectors. The number of trainable layers is a hyperparameter that depends on the amount of training dataset.

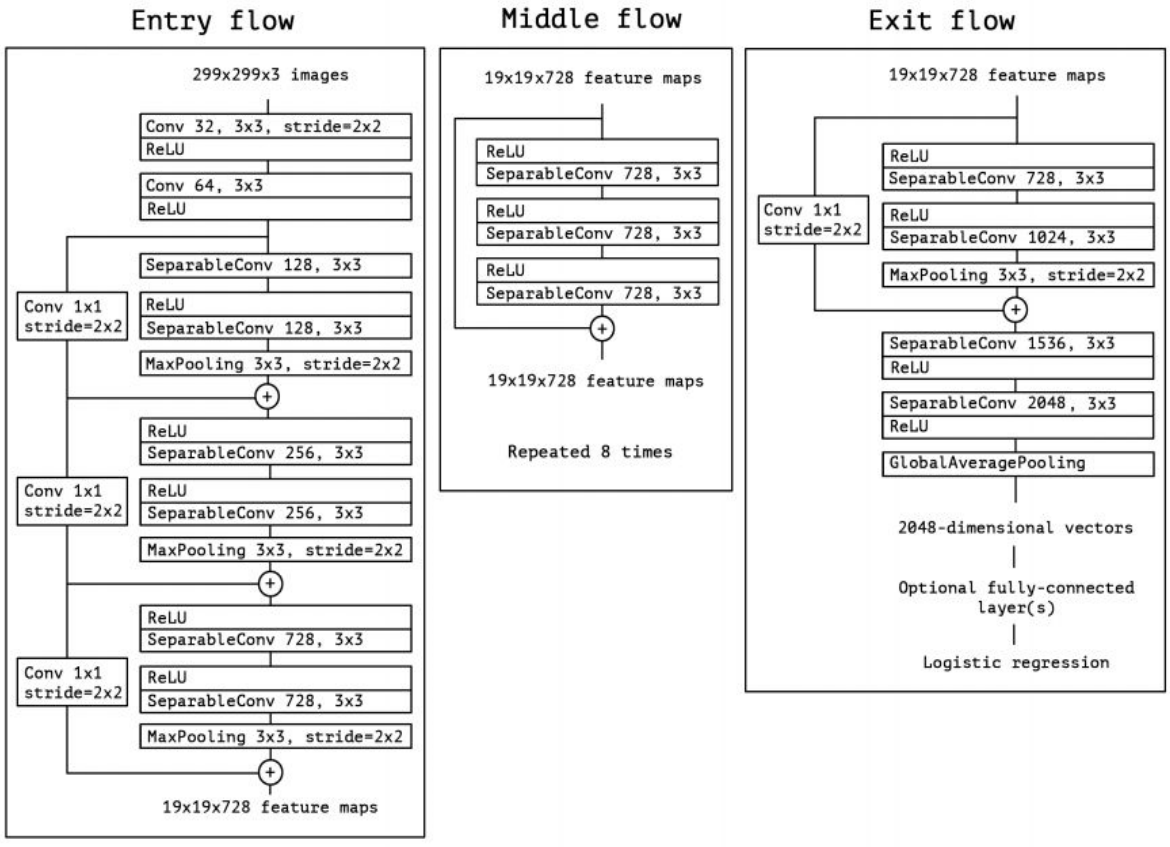


Figure (4)

Then, we added our custom classifier part consisting of LayerNormalization,

Flatten, dropout, dense and softmax layers.

Layer Normalization is used to reduce training time. Dropout layer is used to eliminate some random hidden units' effects to reduce overfitting as shown in figure (5). So after observing an effective overfitting in our results, we used one Dropout layer after the flatten layer and its rate is a hyperparameter that was determined after gradually increasing it until we get the optimum value of 0.6 which is considered a large value but it presents great results in reducing overfitting effect.

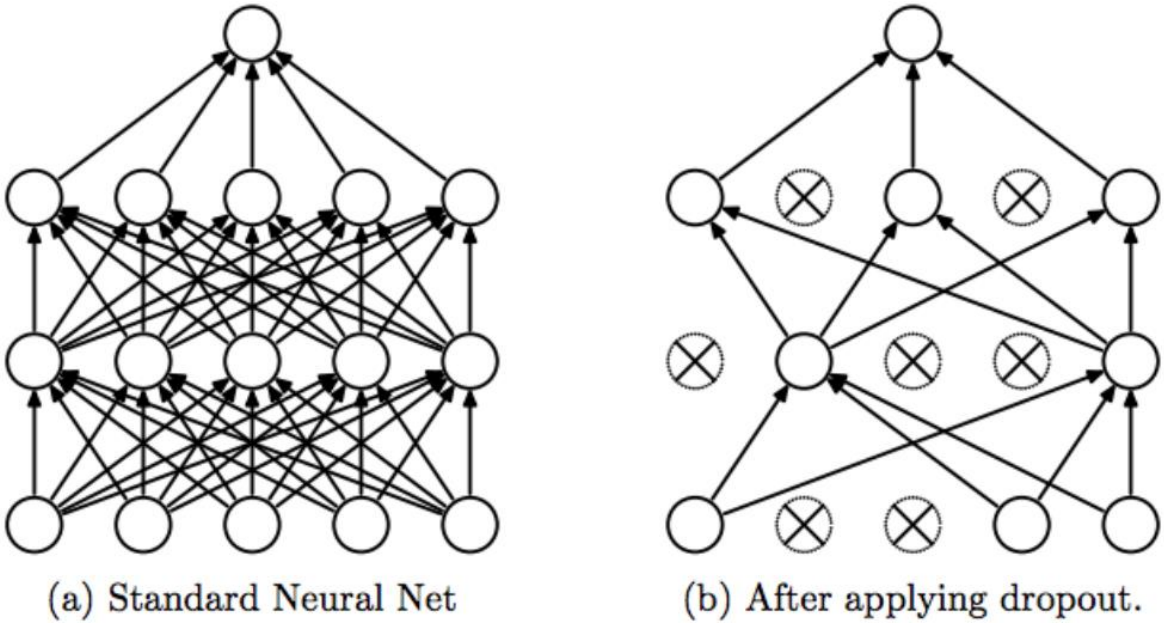


Figure (5)

In Dense layers we should choose the number of hidden units in a way that makes these numbers gradually decay until reaching a suitable value for the number of classes, although in our model there were many limitations in computational resources that limit the process of choosing the number of hidden units.

To determine the number of classes we had two options; the first is to work with 6 classes which is the number of currency classes, and the second option is to work with 12 classes, which represent two faces for each currency class front and back,

it was found that the second option gives better performance so we chose to work with it, and create a simple function to treat the two faces for the same currency as if they were the same thing.

1.5 Optimization Algorithm

Adam optimizer was used for the process of updating the weights of the model's parameters. **Adam,** which stands for adaptive moment estimation, combines the ideas of momentum optimization and RMSProp: just like momentum optimization, it keeps track of an exponentially decaying average of past gradients; and just like RMSProp, it keeps track of an exponentially decaying average of past squared gradients (see Equation)

1. $\mathbf{m} \leftarrow \beta_1 \mathbf{m} - (1 - \beta_1)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
2. $\mathbf{s} \leftarrow \beta_2 \mathbf{s} + (1 - \beta_2)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
3. $\widehat{\mathbf{m}} \leftarrow \dfrac{\mathbf{m}}{1 - \beta_1^{\,t}}$
4. $\widehat{\mathbf{s}} \leftarrow \dfrac{\mathbf{s}}{1 - \beta_2^{\,t}}$
5. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta\, \widehat{\mathbf{m}} \oslash \sqrt{\widehat{\mathbf{s}} + \varepsilon}$

In this equation, t represents the iteration number (starting at 1).

If you just look at steps 1, 2, and 5, you will notice Adam's close similarity to both momentum optimization and RMSProp. The only difference is that step 1 computes an exponentially decaying average rather than an exponentially decaying sum, but these are actually equivalent except for a constant factor (the decaying

average is just 1 – β_1 times the decaying sum). Steps 3 and 4 are somewhat of a technical detail: since m and s are initialized at 0, they will be biased toward 0 at the beginning of training, so these two steps will help boost m and s at the beginning of training.

The momentum decay hyperparameter $\beta_1$is typically initialized to 0.9, while the scaling decay hyperparameter β_2 is often initialized to 0.999. The smoothing term ε is usually initialized to a tiny number such as 10^{-7}. These are the default values for the Adam class.

Since Adam is an adaptive learning rate algorithm (like AdaGrad and RMSProp), it requires less tuning of the learning rate hyperparameter η. You can often use the default value η = 0.001, making Adam even easier to use than Gradient Descent.

1.6 Model Evaluation

The model performance is evaluated by categorical cross-entropy with the following loss function:

$$L(\Theta) = -\sum_{i=1}^{k} y_i \log(\hat{y}_i)$$

Where y is the true output, and y-hat is the predicted output from the model, and k is the number of classes.

1.7 Results

We get a test accuracy of about 90% when our model is tested on 5600 new images that it hadn’t seen before, these are samples of the results:

خمسون جنيها

عشرة جنيهات

مائة جنيه

عشرون جنيها

مائتان جنيه

خمسة جنيهات

1.8 Future work

As a result of unavailability of good computational resources, our model didn’t get perfect results because we had some limitations in our training process, so we hope in the future we get better resources to improve our model performance.

Also, as a result of unavailability of Egyptian currency dataset we had some limitations in accuracy as our generated dataset is not good enough, we hope it will be available soon.

Finally, we target to use this model with all Arab currencies as soon as we get dataset for any of them.

1.9 References

1-Currency Recognition System for Visually Impaired: Egyptian Banknote as a Study Case

2- [2017 CVPR] [Xception] Xception: Deep Learning with Depthwise Separable Convolutions

3- Dropout in (Deep) Machine learning

4- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd edition, Aurélien Géron.

2. Arabic Image Captioning

2.1 Abstract

What can you see in this image?

Someone can say he sees a "**A man riding a motorcycle" or "A building like a stadium" or "A car in front of a big tree"**

All these descriptions are related to the image and this task is easy for humans as we have a very effective and complicated visual system, but what about machines? Is this an easy task for them? Can we write a program which takes a photo and produces a caption relevant to it in English or Arabic or any language?

For computers, it's pretty challenging, but yes, the revolution which deep learning made in the past few years doesn't make this task difficult and can be solved if we have the required dataset

This problem is researched well by Andrej Karapathy in his PhD at Stanford who is now the Director of AI at Tesla

We are in the century of data “Data is the new electricity” Andrew NG said

For example, around 300 million pictures are uploaded each day to Facebook (Inc., 2018). As image captioning needs to extract information about the content of the image and express it in a readable sentence, models need to achieve several objectives including object detection, extraction of relationships among objects, etc.

Why Arabic image captioning?

Consideration for the Arabic language is important since it is spoken by more than 422 million people around the world, and it is the native language in 22 countries. Arabic is also ranked the fourth mostly used language on the web. We can see that most technologies In this field is designed for English or Chinese speakers, so we decided to make a new thing and serve Arabic speakers in our country Egypt or in the Arab world despite of the challenges we expected to face such as the scarcity of Arabic resources and difficulty of Arabic language as it has over 12 million words compared with English which has approximately 1 million words.

2.2 Dataset

There are various datasets for image captioning, the datasets are Fliker8k, Fliker30k and MACOCO. Fliker8K contains 8k images, the images in this dataset mainly contain humans and animals, and every image has approximately 5 annotations. Fliker30k images has 31783 annotations and it is extended from the Flickr 8k dataset.

MSCOCO dataset contains 82,783 training images with 414,112 annotations and 40,504 validation images with 202,653 annotations, this data is pretty big and we saw also in some surveys that it gives better results than others, so we decided to use it. First, we translated annotations using google translation API and we wished to edit and validate it by professional Arab translators but unfortunately, we don’t

have fund for that so we considered this process as a future enhancement. Let’s go on how we divide our dataset, there is a famous approach which called karpathy splits, it adds training and validation data to each other, this split contains 113,287 training images with five captions each, and 5,000 images for validation and testing and it is used in most research papers for comparing the results and as this split is for English captions, we generated a simple python script in order to make our Arabic data split the same way.

2.3 Captions Preprocessing

2.3.1 Data Cleaning

After we prepared our data splits, it’s time to clean it, so what does cleaning mean?

Cleaning in NLP science is to remove some words or characters which don’t improve the model accuracy but instead affect it negatively such as

punctuations and diacritics (Tashdid, Fatha, Damma, etc..), normalize "همزة" and "هاء", remove English characters, and we can see an example and see that if we don’t make the cleaning process the size of the vocabulary can be big as (سيارة و سياره) are considered as two different words and this will increase make model training very slow, so it's a must to clean them as we don’t target a very correct Arabic spelling output.

2.3.2 Tokenization

Tokenization is a way of separating a piece of text into smaller units called tokens. Here, tokens can be either words, characters, or sub words. Hence, tokenization can be broadly classified into 3 types – word, character, and sub word (n-gram characters) tokenization.

For example, if we want to tokenize " رجل يقف في الشارع "it will be

[رجل, يقف, في, الشارع] , but we know that our models don't understand words, it needs numbers, so after tokenization, we create a vocabulary of the most common words in our data and in our situation we have a vocabulary of 50,006 words, each word in the vocabulary takes an index from 0 to 50,006 and then we convert our tokens to numbers in order to be able to train it, and if a word is not found in the vocabulary we can replace it by <UNK> to represent undefined words, but wait, we have various length captions, does the model accept lists of tokens with various lengths?

No, we choose the maximum length from all sentences and consider it as our caption input length, and if we have captions less then this length we pad them until they reach the maximum length value, here we choose the length to be 20, an important note is to add <start> and <end> words to our caption to make the LSTM Know when our caption starts and ends.

So, our final token will be like this:

[1, 4, 3, 100, 1234, 2, 0, 0, 0, 0, 0, 0, 0, 0,0 ,0 ,0 ,0 ,0 ,0]

Where 1 corresponds to the <start> token, and 2 corresponds to the <end> token.

2.3.3 Embedding

Word embeddings are a type of word representation that allows words with similar meanings to be represented in a similar way, each word is mapped to one vector and the vector values are learned in a way that resembles a neural network.

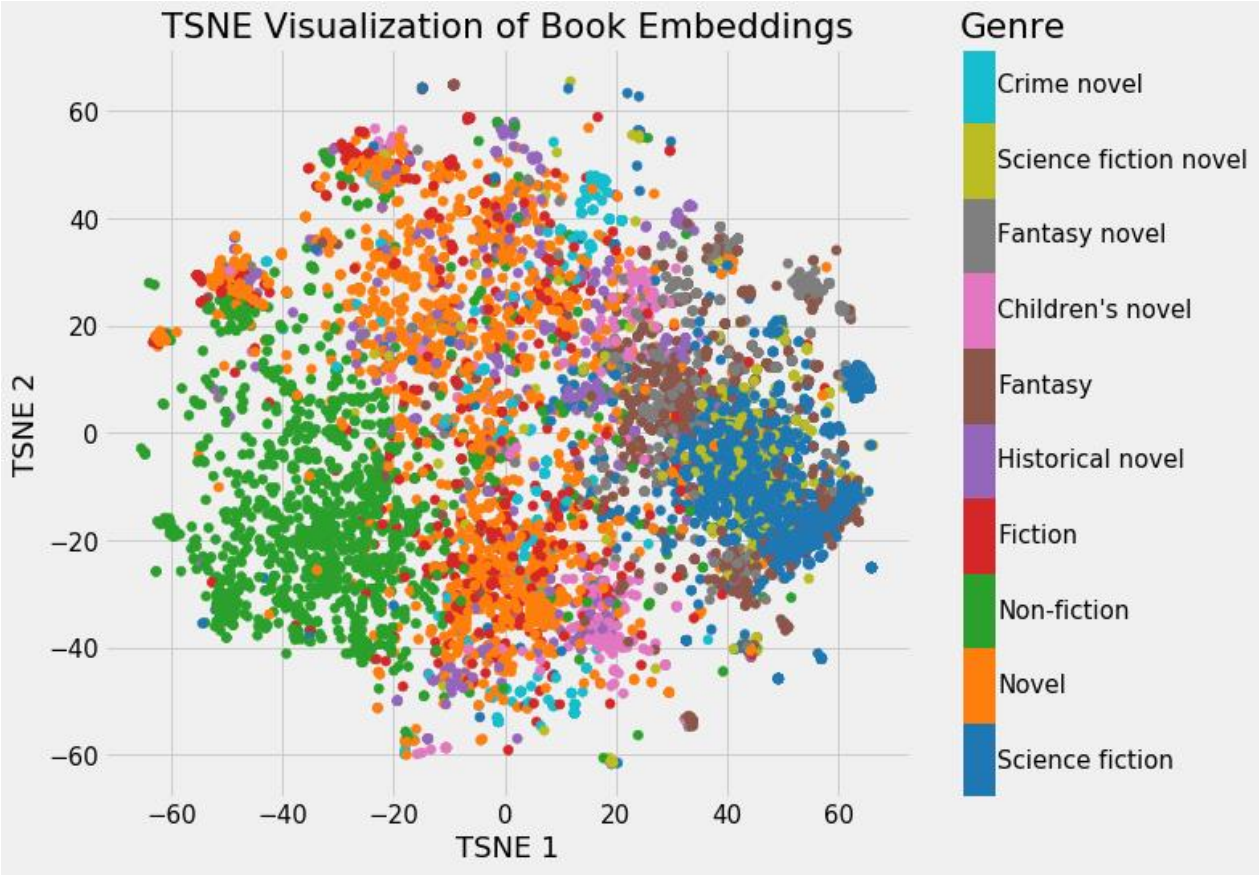
We have two approaches to get embeddings:

1. Learn embedding:
   Where a model is trained to learn the embedding on its own and be used as a part of another model for a task later. This is a good approach if you want to use the same embedding in multiple models but this needs a huge computational resources and time.
2. Use a pre-trained model:
   It is common to use this approach instead of training our word embedding from scratch and there is a famous English word embedding like word2vec and GloVe.

For Arabic word embedding, it is a bit challenging as there is a lack of data and resources in Arabic, but we have two options:

1- **AraVec:** it is a pre-trained, distributed word representation (word embedding), open source project that aims to provide the Arabic NLP research community with free and powerful word embedding models. AraVec has been published in the 3rd International Conference on Arabic Computational Linguistics (ACLing 2017), Dubai, UAE, 2017. It uses CBOW on a dataset collected from Twitter and it produces a 300-dimentional vector.
2- **Universal-multilingual-sentence-encoder:** Developed by researchers at Google, 2019, v2 [1]. Convolutional Neural Net. Covers 16 languages, showing strong performance on cross-lingual retrieval. The input to the model is a variable length text and the output is a 512-dimensional vector.

We used both, but the universal-sentence-encoder gave better accuracy. And the below figure is an example of how word embedding control words of similar meanings:



2.4 Approach

We follow a technique called bottom-up-top-down attention, the caption model takes as input an image and to produce the caption, it encodes the image to its feature vector such that each image feature indicates a prominent region of the image.

2.4.1 Feature extraction (Bottom-up attention or image model):

To know what is in the image and describe it, we need to know various objects in the image and to know that we need to train the image to make object detection and recognition, but this parallel to the language model needs huge computational resources and will be very expensive so we intend only to train the language model. So how can we see what is inside the images?

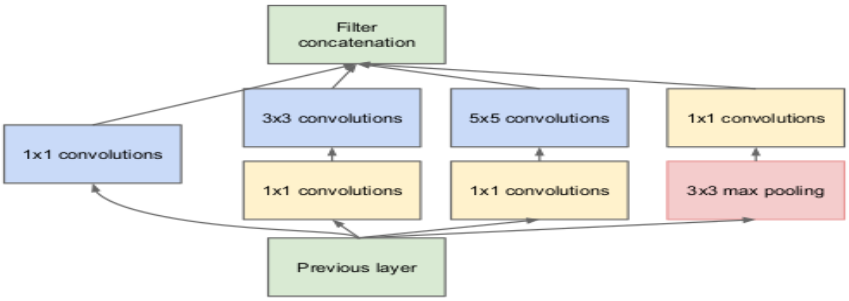
The deep learning community is very helpful, if someone makes a successful training, they make the project open source with the pretrained weights in order for people to use it instead of retrain from scratch, so we benefit from this advantage. This concept is called transfer learning.

To get features we used InceptionV3 model pretrained on the famous imagenet dataset from Keras applications, there are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning

This model takes an image of size (299x299 x3) after making some preprocessing on it and returns a feature matrix of size (64x2048) which returned from the layer before last.

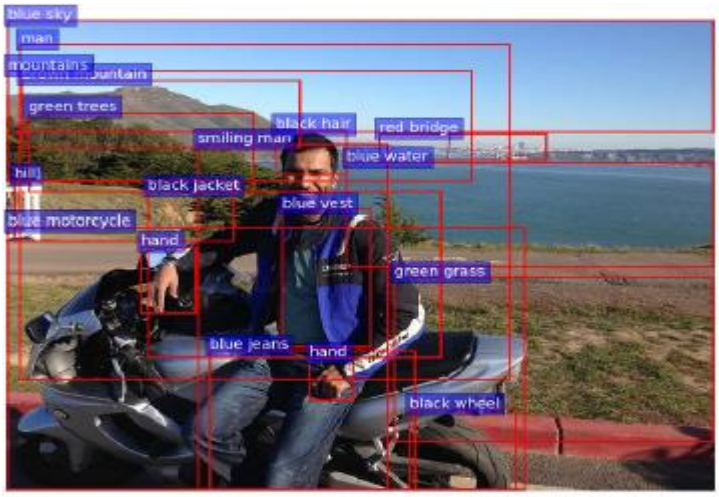
convolution
max pool
convolution
max pool
inception (3a)
inception (3b)
max pool
inception (4a)
inception (4b)
inception (4c)
inception (4d)
inception (4e)
max pool
inception (5a)
inception (5b)
avg pool
dropout (40%)
linear
softmax

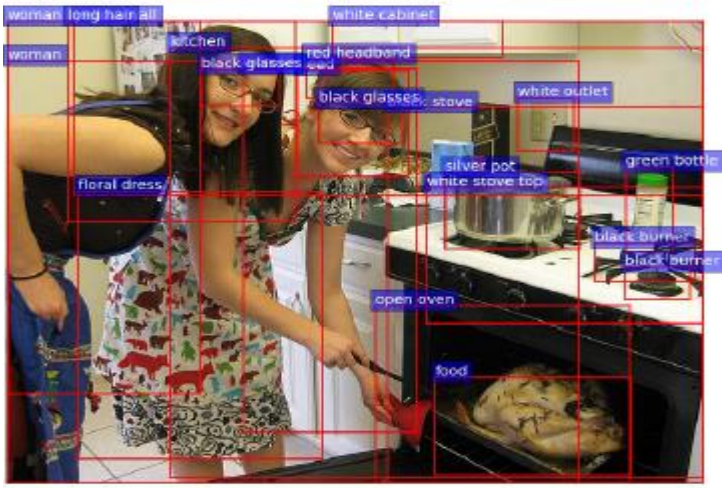
InceptionV3 model summary



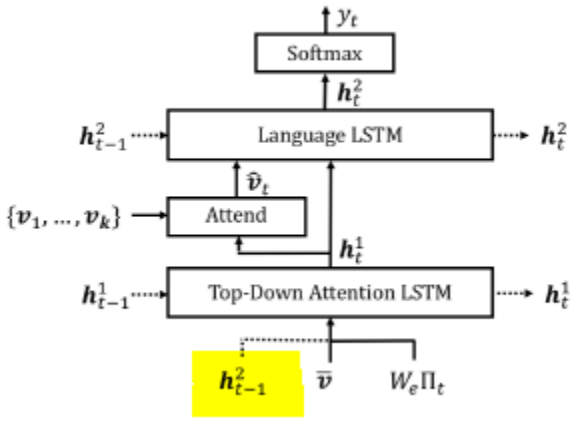
"neurons that fire together, wire together"

And this is also how the model works in object detection and how we take the features of each box in the image:





2.4.2 Language model (Top-Down Attention LSTM)



As we see, we have two LSTM layers:

1. Top-Down Attention LSTM:
   The input vector to the attention LSTM at each time step consists of the mean-pooled image feature $\bar{v} = \frac{1}{k}\sum_i vi$ concatenated with an encoding of the previously generated word, given by: $x_t^1 = [\ \bar{v}\ , W_e \prod_t\ ]$.
   Where $W_e \in R^{E\times|\Sigma|}$ is a word embedding matrix for a vocabulary Σ, and $\prod_t$ is one-hot encoding of the input word at timestep t. These inputs provide the attention LSTM with maximum context, the overall content of the image, and the partial caption output generated so far respectively. The word embedding is learned on top of pretrained weights.

   Given output h_t^1 of the attention LSTM, at each time step t we generate a normalized attention weight $\alpha_{i,t}$ for each of the k image features v_i as follows:

   $$a_{i,t} = w_a^T \tanh(W_{va}\, v_i + W_{ha}\, h_t^1)$$
   $$\alpha_{i,t} = \text{softmax}\ (a_t)$$

   where $W_{va} \in R^{H\times V}$, $W_{ha} \in R^{H\times M}$ and $w_a \in R^H$ are learned parameters. The attended image feature used as input to the language LSTM is calculated as a convex combination of all input features:

   $$\hat{v} = \sum_{i=1}^{k} v_i\, \alpha_{i,t}\, v_i$$

2. Language LSTM:
   The input to the language model LSTM consists of the attended image feature, concatenated with the output of the attention LSTM, given by: $x_t^2 = [\ \hat{v}_t, h_t^1]$ and then the output h_t^2 fed to a softmax layer to generate the final output.

   The model performance evaluated by categorical cross-entropy with the following loss function:

$$L(\Theta) = -\sum_{i=1}^{k} y_i \log(\hat{y}_i)$$

   Where y is the true output, and y-hat is the predicted output from the model, and k is the number of classes.

2.5 Training

The model is trained with Adam optimizer which can be used instead of the classical stochastic gradient descent procedure to update network weights iteratively based on training data and with configuration parameters as follow:

- **alpha**: Also referred to as the learning rate or step size. The proportion that weights are updated (0.001).
- **beta1**: The exponential decay rate for the first moment estimates (0.9).
- **beta2**: The exponential decay rate for the second-moment estimates (0.999).
- **epsilon**. Is a very small number to prevent any division by zero in the implementation (1e-07).

Mini-batch learning technique is used for training this model by building a data generator that takes only 128 samples at the time and shuffles them. The Mini-batch technique is mandatory because the size of data is so large that it can't possibly fit in a computer with a plausible RAM size.

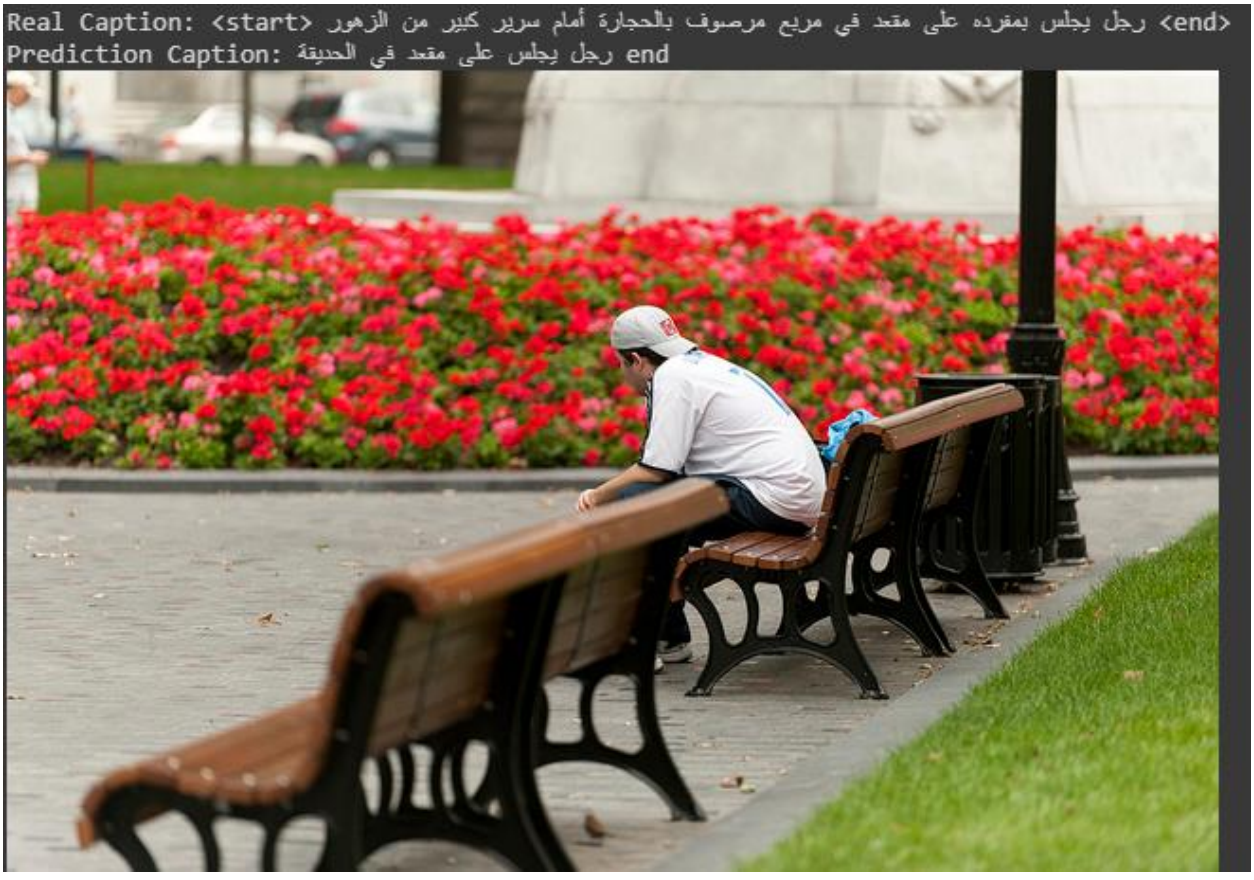
The resource used to experiment and obtain the best hyper parameters for this model is Google Colab Pro, which provides a high RAM as well as GPU computing capabilities and is much faster than regular computers.

2.6 Evaluation and Results

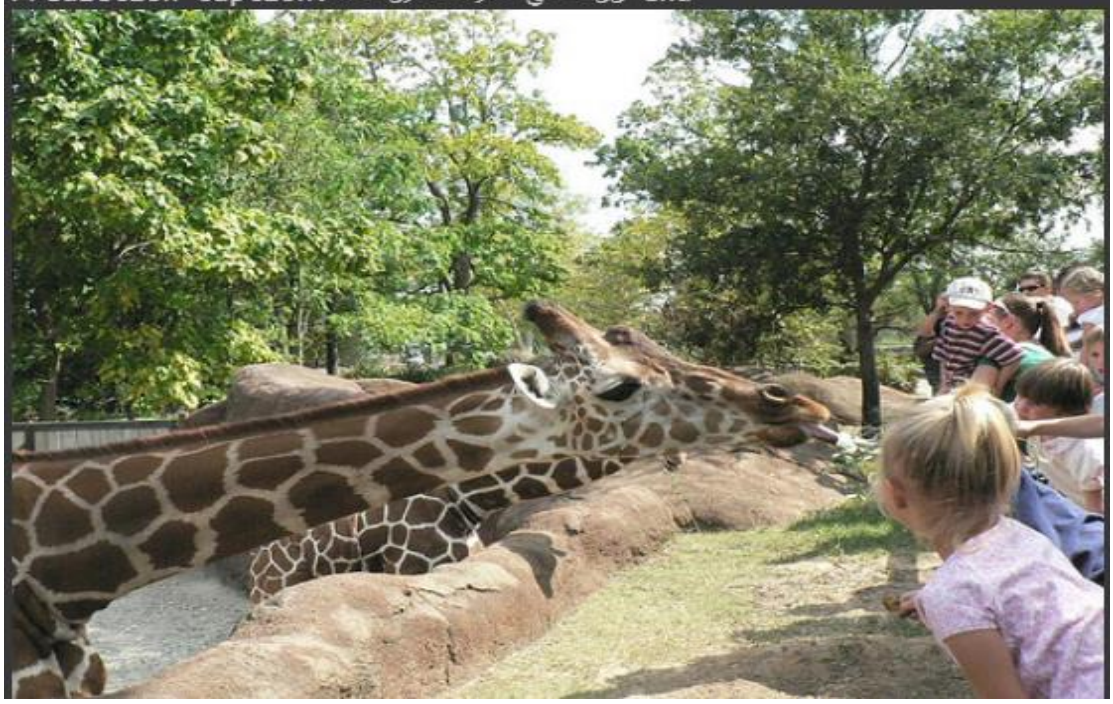
To evaluate caption quality, we use the COCO API evaluation tool which generates the results of the standard automatic evaluation metrics and 5,000 test images from karpathy splits, and we get above 12% BLEU-4 which exceeds the current work in Arabic captioning, and we look forward to achieving better results after doing further experiments.

Below are some results for our Arabic captioning model:


Prediction Caption: دراجة نارية متوقفة على جانب الطريق end
LML
HALAWA

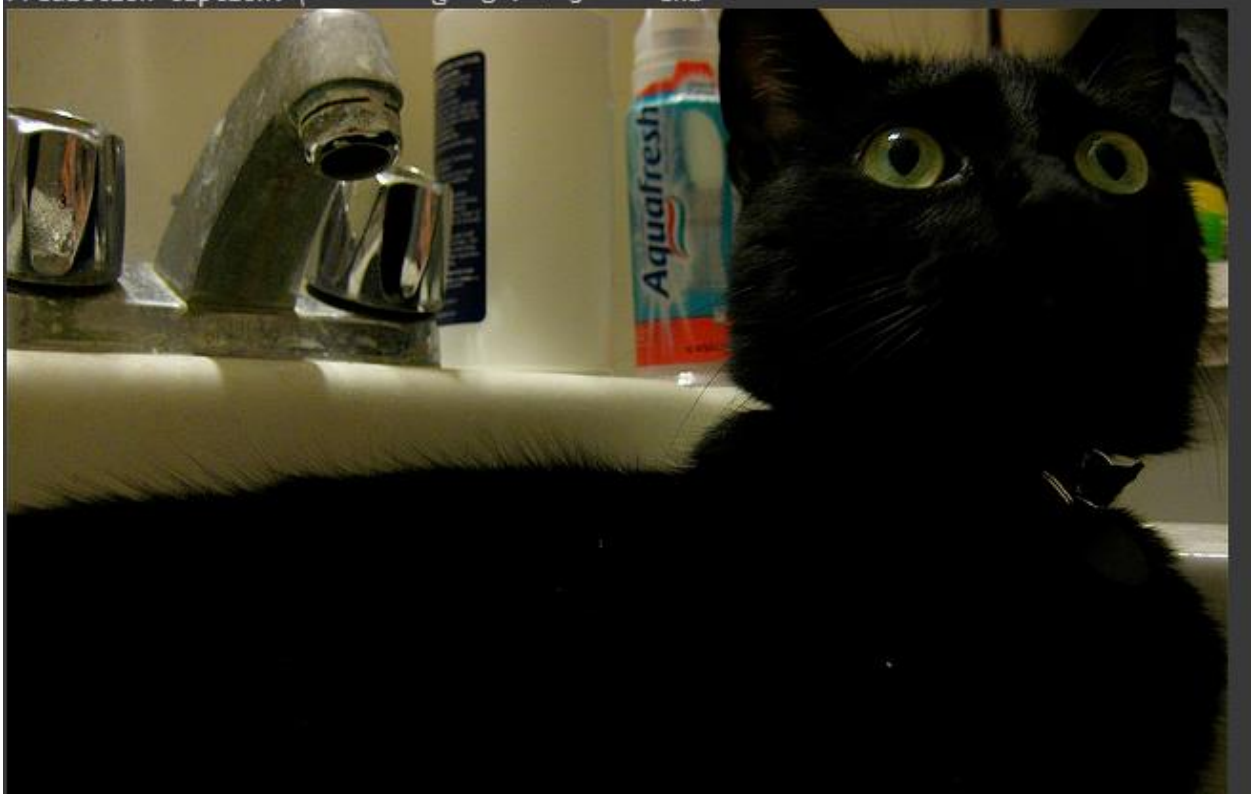

Real Caption: <start> رجل يجلس بمفرده على مقعد في مربع مرصوف بالحجارة أمام سرير كبير من الزهور <end>
Prediction Caption: رجل يجلس على مقعد في الحديقة end

Real Caption: <start> مجموعة من الأطفال يقومون بإطعام حيوان. <end>
Prediction Caption: زرافة في حديقة للحيوانات end



Real Caption: <start> رجل يركب دراجة نارية زرقاء بالخارج <end>
Prediction Caption: شخص يركب دراجة نارية end

Real Caption: <start> تَوجد قطة سوداء في حوض الحمام لتتوافق بشكل مثالي. <end>
Prediction Caption: قطة سوداء تجلس على مغسلة الحمام end



Real Caption: <start> تَم إعداد كمبيوتر محمول على طاولة غرفة الطعام. <end>
Prediction Caption: كمبيوتر محمول مفتوح يجلس على طاولة مع كمبيوتر محمول مفتوح يجلس على طاولة مع كمبيوتر

2.7 Future Work

We need to enhance the results of the model so we need to train the model on a good GPU Machine to enhance object detection and therefore it can detect more objects and give us better captions.

We have some faulty results which give us wrong words not in the scene or repeat some words until the end of caption length like the last figure above, and this problem was solved in a paper in CVPR 2020 conference so we will work on it.

2.8 References

- Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering
- Show and Tell: Lessons learned from the 2015 MSCOCO Image Captioning Challenge
- VQA COCO Dataset
- AraVec Embedding
- InceptionV3 Image Model
- Google Colab
- Adam Optimization
- Cross Entropy

3. Visual Question Answering

3.1 Abstract

Deep learning has had a revolutionary impact on computer vision since deep neural networks come to light again in the past two decades. There remain a lot of areas that are challenging and ongoing research is major to its advancement, Visual Question Answering is one of them.

Question: ماذا يفعل الرجل في الشارع؟
Answer: المشي

Question: على ماذا تقف الفتاة؟
Answer: لوح تزلج

Question: ما نوع السيارة المعروضة؟
Answer: حافلة

Question: كم عدد المراكب الشراعية في الخلفية؟
Answer: 1

A Visual Question Answering model takes an image and a question as input and gives a short answer as output. Most of the attempts have been building VQA models for the English language only, since this project is for Arabic speakers, the VQA built here is built with Arabic data which proposed a bigger challenge, but it had produced very promising results.

3.2 Dataset

The data used to train the model is COCO open-ended questions dataset of balanced real images. Training data consists of 82,783 images, 443,757 questions and 4,437,570 answers. Validation data consists of 40,504 images, 214,354 questions and 2,143,540 answers.

Originally, the questions and answers were in English, with multiple questions for each image and multiple viable answers for each question.

The original training and annotation (answers) files were in JSON format with the following parameters for questions: “image_id”, “question_id”, “question”, and the following parameters for answers: “question_type”, “answers”: {“answer”, “answer_confidence”, “answer_id”}, “image_id”, “answer_type”, “question_id”.

A processing was made to assort the data in an excel file with three columns, “questions”, “answers”, “images”, with answers only holding the highest repeated answer in the annotation data, and images holding the image ID.

Then this excel file was translated by Google Translate from English to Arabic. The resulting excel file is well organized and easy to use for training.

3.3 Question Embedding

The difference between building a VQA model for English or Arabic is only seen in the embedding for question words, which is converting each word into a 100,

300, or 500 floating-number-vector representing the features of this word.

Word embedding is trained by a variety of language models like Skip-Grams or Continuous Bag-Of-Words (CBOW) and it takes a lot of time to train but the result is general among any use for the language, that's why a pre-trained word embedding for the Arabic language is used here.

Google global sentence encoder was first used on the model, it produces a 500-dimentional vector for each word, but then another embedding called AraVec had produced better results and proven to be a better option. AraVec has been published in the 3rd International Conference on Arabic Computational Linguistics (ACLing 2017), Dubai, UAE, 2017. It used CBOW on a dataset collected from Twitter and it produces a 300-dimentional vector.

For using AraVec, a preprocessing on the questions is required first to clean the input string like removing repeated characters and replacing {"آ" ،"إ" ،"أ"} with "ا", replacing all "ى" with "ي", removing tashkeel and punctuation marks. This processing has an additional advantage in which it minimizes the error that could result from different spelling when testing.

A function *clean_str* is built and performed on question data only and new input is passed through it when testing.

After loading the pre-trained embedding, an embedding matrix is built for the words that occur in the training data only, first by tokenizing the questions data, i.e. converting each word into a corresponding number based on the frequency of the word, more frequent word takes the number 1, then 2, and so on, in addition to an <*unk*> token to represent words not in the embedding model and in testing, new words that weren't in the training data.

After tokenizing the questions, all sentences are converted by the tokenizer and fixed at a specific length (15) with zero padding, the tokenizer is built with Keras and is saved for later processing on new input.

As an example, the sentence:

ما هو لون قبعة الرجل؟

Is converted to:

[3, 5, 11, 116, 15, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

3.4 Output Method

The required output for the VQA model, i.e. a short answer for each question, is built as a classification job, in which there is a fixed number of answers, whether it's a single or multiple words, these answers are built by choosing the 1000 most frequent answers in the data and training only samples which have answers in this range, a method of trying 1500 was also tried.

This method has proven to be more effective than including all answers in the output, because it considerably reduces the output classes without a large reduction in training samples (about 30,000 samples from 440,000 samples only weren't included).

The final 1000 answers are also tokenized, each answer corresponds to a number, and the file saving the correspondence of this tokenization is saved for later testing.

By this method of creating the output, it can be easily separated for further testing on specific type of answers like yes/no questions for example or numeric answers. For example, experiments have shown that VQA models or image features in general can't comprehend object counting, so a filtering was made to replace every numeric answer related to a counting question larger than 3 with the word "كثير".

3.5 Image Features

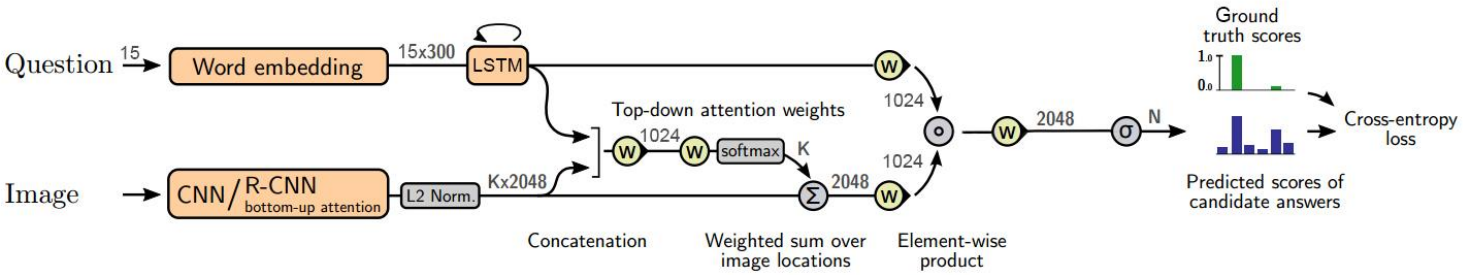
Extracting features from images by a convolutional neural network model takes a lot of time and requires expensive resources. Thus, many deep learning models resort to pre-trained image features model.

For VQA, three image models were tried; Inception V3, VGG19, and Xception model. While VGG19 takes considerably less time in training the whole model, Inception and Xception models produced better results on external images, the Xception model was chosen for the final architecture, and in the code, these models are loaded by Keras.

The difference between image models and the inner implementation of the Xception model was previously discussed in section (1.4.3).

3.6 The Model

After processing all the data in the appropriate format, the next step is describing the model by which it is trained, below is a figure summarizing the model architecture:



Yellow circles represent linear layers with learnt weights. The number after each layer indicates its output shape. K is 100 for the Xception image feature model.

The model takes two inputs, a 15-dimentional tokenized question and an image. The feature of the image is extracted using Xception's bottom-up attention model and normalized to produce a 100x2048 feature matrix.

The question is processed with the embedding matrix to produce a 15x300 linguistic feature matrix. This matrix is concatenated with the image features and passed through a linear layer with ReLU activation (Rectified Linear Unit) and a size of 1024 hidden units, then another layer with softmax activation with 100 hidden units. A dot-product is applied between the result of the softmax and the image feature matrix, (2048x100) · (100x1) = (2048x1) dimentional layer, this layer is passed through another linear layer of 1024 hidden units with 0.3 Dropout regularization.

The language feature matrix is passed through an LSTM layer of 1024 hidden units, this layer is multiplied by the 1024 layer resulting from the image feature

processing and is the fusion of the two inputs, this layer passed through a linear layer with 2048 hidden units with 0.5 Dropout regularization, and the final layer has a dimention of N which is the output size (1000 for this case) and it has softmax activation.

The model performance evaluated by categorical cross-entropy with the following loss function:

$$L(\Theta) = -\sum_{i=1}^{k} y_i \log(\hat{y}_i)$$

Where y is the true output, and y-hat is the predicted output from the model, and k is the number of classes.

3.7 Training

The model is trained with Adamax optimizer. It is a variant from Adam optimizer and it works by storing an exponentially decaying average (v_t) of past gradients (m_t) with its three parameters, learning rate, β_1, β_2. The optimizer's job is to update the learned gradients at each step with a proper learning rate to speed it up at the beginning and slow down when it comes close to a minimum value for the loss function.

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\ \hat{v}_t &= \max(\hat{v}_{t-1}, v_t) \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} m_t \end{aligned}$$

Adamax optimizer equation, where t is the iteration number, and θ is the learnable weights.

Mini-batch learning technique is used for training this model by building a data generator that takes only 64 samples at the time and shuffles them. The Mini-batch technique is mandatory because the size of data is so large that it can't possibly fit in a computer with a plausible RAM size.

The resource used to experiment and obtain the best hyper parameters for this model is Google Colab Pro, which provides a high RAM as well as GPU computing capabilities and is much faster than regular computers. After finding the best hyperparameters, the whole dataset was trained on a 16GB-RAM Core i-5 7th Generation Laptop.

Various tries have been made to obtain the best options to be used on the model. Below is a snapshot of the sheet that was used to document the tries:

Training Samples No.	Validation Samples No.	OutputDim	Model change #1	Model change #2	Model change #3	val_accuracy	train_accuracy	General Notes:
130k	30k	1000				40%	49%	Standard accuracy after epoch 10
360k	30k	1000				44%	51%	
100k	20k	1000				40%	46%	
100k	20k	2000				34% epoch 5		Doesn't converge easily
130k	10k	1000	Added dropout 0.5 after add layer			42%	50%	
130k	10k	1000	Embedding trainable=False	Added dropout 0.5 after add layer		41%	41%	
120k	20k	1000	Added dropout 0.5 after add layer	elu activation and he kernel initializer in Dense layers		41.89%	45.63%	
120k	20k	1000	Added BatchNormalization after add layer	leaky relu activation and he kernel initializer in Dense layers		37%	62%	
120k	20k	1000	Batch after image, leaky relu activation	drop after add, drop after mul	hidden units = 2048	40%	40%	No batch, dropout 0.2 -> train_acc = 49%
110k	30k	1000	vgg16 image features	drop after add	extra dense before output, l2 regularization	40.65	42.83	Over many failed experiments, it was useless to increase hidden units.
250k	30k	1000	vgg16 image features		extra dense before output, l2 regularization	42.67	51	So much faster
360k	30k		vgg19 image features	l2 regularization	drop 0.3 after add	44.15, 11 epoch	47.65	
281k	20k	977	inception	Original model		43	41	13 min / epoch
360k	30k	1000	vgg19	aravec embedding	drop 0.3 after add layer	46	51	
360k	30k	1000	vgg19	aravec embedding	no high count	49.7	54	
370k	30k	1500	vgg19, aravec	drop 0.3	no high count	48	55	
312k	78k	1000	vgg19, aravec	old model		44	50	15 epoch
180k	30k	1000	xception, aravec	drop 0.3 after fc1, add		44	54	

3.8 Results

The representation of the last layer of the model (of size N which is the number of answer classes, 1000 for this case) holds the probability that the answer to the question and image is of the current index, the ground truth data has a 1 in the index that is corresponding to the right answer and 0 elsewhere.

Evaluation is made by accuracy calculation on the sum of difference between predicted and true values of the classes. The best model for VQA produced an accuracy of 49% on validation data which is the better recorded accuracy for Arabic Visual Question Answering.

Answers are ordered by their probability and it is usually a good indication of how confident the model is about the result. Although it has a 49% accuracy, in many cases the right answer is apparent in the second or third choice.

Below are some results of the final trained VQA model on images it hadn't seen before:

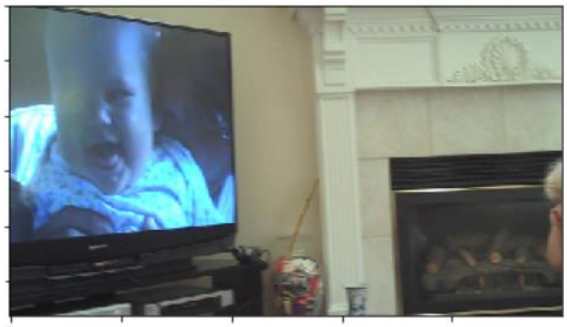
Question: ماذا في الخلفية؟

Probability:	Answer:
0.42192116	الأشجار
0.054707866	نجيل
0.045699153	الجبال

Question: ما لون الحافلة؟

Probability:	Answer:
0.26529184	ازرق وابيض
0.22641745	أزرق
0.14810365	أبيض

Question: في أي عام خرج التلفزيون؟
Answer: 2008



Question: لماذا تستلقي البقرة؟
Answer: متعبه

3.9 Future Work

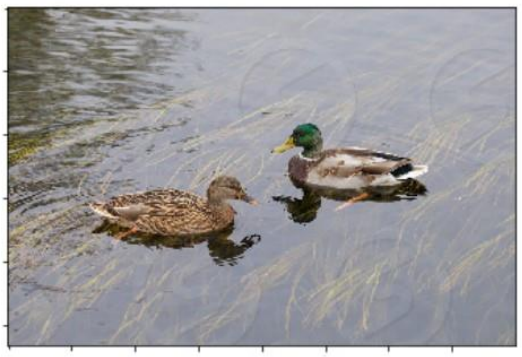
Some limitations of the results of the current model is due to the COCO dataset being generally for open-ended questions not simple questions about pictures that a visually-impaired person for example might need to use. An example of the majority of COCO questions is as follows:

Question: هل هذه الصورة أبيض وأسود؟

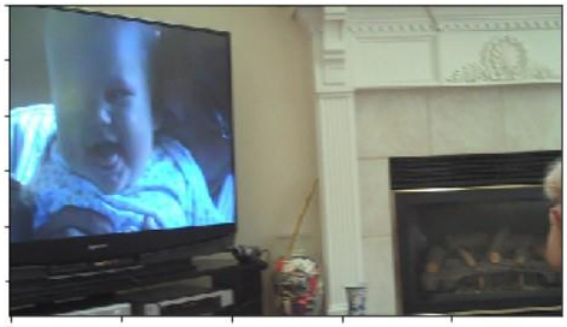
Probability:	Answer:
0.9250391	نعم
0.07495908	لا

Question: ما هذا الحيوان؟

Probability:	Answer:
0.8210042	بطة
0.12202416	طائر
0.026625788	بشري



Question: في أي عام خرج التلفزيون؟
Answer: 2008



Question: لماذا تستلقي البقرة؟
Answer: متعبه

Answers of this kind are implausible for a deep learning model to conclude from image features. Thus, it makes the model fixate on the question type only. But as the model has produced good results despite this fallacy in data, it can produce considerably better results when trained on more plausible data. So, an attempt to collect data of this kind will be made in the future.

Ongoing research in loss functions and image features extraction models will indeed contribute hugely to deep learning models like VQA. And experiments with language processing techniques that are directed to the Arabic language like reducing each word to its root can also be tested.

An approach to format the data so that there is more than one viable answer, or each answer having a value of its probability to be the answer could also be tried.

3.10 References

- Tips and Tricks for Visual Question Answering: Learnings from the 2017 Challenge
- VQA COCO Dataset
- AraVec Embedding
- Xception Image Model
- Google Colab
- Adam Optimization
- Cross Entropy

4. Emotion Recognition

4.1 Abstract

Human emotions, as described by some theorists, are discreet and consistent reactions to internal or external events of importance to the organism. We constitute a big part of our non-verbal communication. Among human emotions, happy, sad, fear, rage, shock, disgust and neutral are the seven core emotions. Facial gestures are the safest way to convey feelings. In this time of booming human-computer interaction, allowing computers to understand these emotions is a crucial challenge. Each facial expression has an amalgamation of emotions. In this paper, we defined the different emotions and their level of strength in the human face by implementing a deep learning approach through our proposed Convolution Neural Network (CNN). The design and algorithm here produce appreciable results that can be used as an inspiration for further work in a computer-based emotional recognition system. We achieved an accuracy of 67% on FER2013 test dataset.

4.2 Related Work

Many of the important contributions made in this field are facial expression recognition based on local binary patterns [3]. Emotion recognition using the binary decision tree [4], Facial Expression Recognition using Convolutional Neural Networks [5]. Modular Self spaces system for classifying emotions using NN and HMM [6], Emotion interpretation in visual and audio cues [7], Combining various kernel methods [8]. Yet these computational approaches are far behind human precision, since their basis is not focused on the functionality of human deep learning and training. The objective of our study is to analyze facial emotions in static images using the different attempts of the Convolutional Neural Network (CNN). CNN [9] is a special kind of deep learning approach that offers solutions to many of the problems of image recognition after intensive training. Due to the lack of a sufficient amount of experience, it is difficult for humans to identify emotions in the face. For example, we can't completely decide whether a person is shocked or pleased. We are therefore trying to explore the matter and to examine the different degree of emotions present in the human face at one case.

4.3 Dataset

FER2013 dataset has been used for experiment. FER2013 is an open-source dataset which is first created for an ongoing project by Pierre-Luc Carrier and Aaron Courville, then shared publicly for a Kaggle competition, shortly before ICML 2013.

4.3.1 Dataset Overview

This dataset consists of 35.887 grayscale, 48x48 sized face images with various emotions -7 emotions, all labeled-.

Emotion labels in the dataset:

0: -4593 images- Angry

1: -547 images- Disgust

2: -5121 images- Fear

3: -8989 images- Happy

4: -6077 images- Sad

5: -4002 images- Surprise

6: -6198 images- Neutral

During the competition, 28.709 images and 3.589 images were shared with the participants as training and public test sets respectively and the remaining 3.589 images were kept as private test set to find the winner of the competition. The dataset was set to be accessible to everyone after completing the competition.

4.3.2 Dataset Preprocessing

As realized at first glance, the images in the Excel file are stored with the corresponding pixel values on each row and preprocessing on the data is required - a little bit:

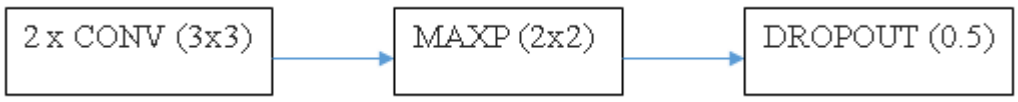
- Converting the relevant column element into a list for each row.

- Splitting the string by space character as a list.
- Normalizing the image.
- Resizing the image.
- Expanding the dimensions of channel for each image.
- Converting the labels to categorical matrix.

4.4 Model

Now the training dataset is available and it is time to train the model. In this section, we present our proposed CNN model architecture.

The main module in the model architecture is shown below:



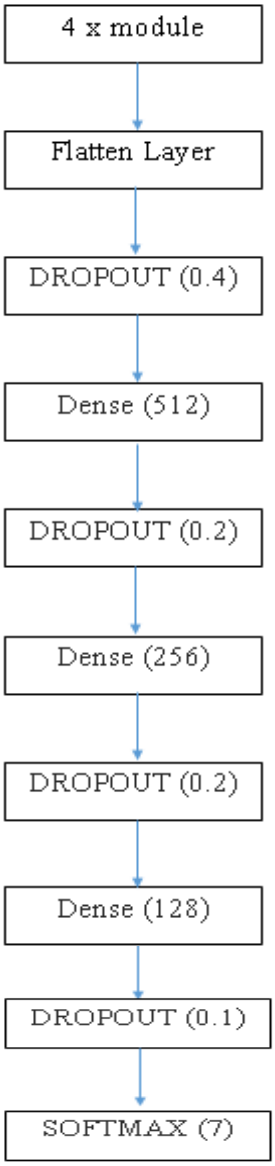
- In the first convolutional layer, L2 regularization (0.01) has been added.
- In all convolutional layers except the first one, batch normalization layer has been added.
- MAXP (2x2) and DROPOUT (0.5) layers have been added to each convolutional layer's block.
- "RELU" has been picked as activation function for all convolutional layers.

This module has been used 4 times in our proposed model to extract features from images, the feature vector dimension in our proposed model is (64x4x512) which is then passed to the classifier part.

This table shows the number of parameters.

Total parameters	5,905,863
trainable	5,902,151
Non-trainable	3,712

The whole architecture of the model is shown in the following figure:



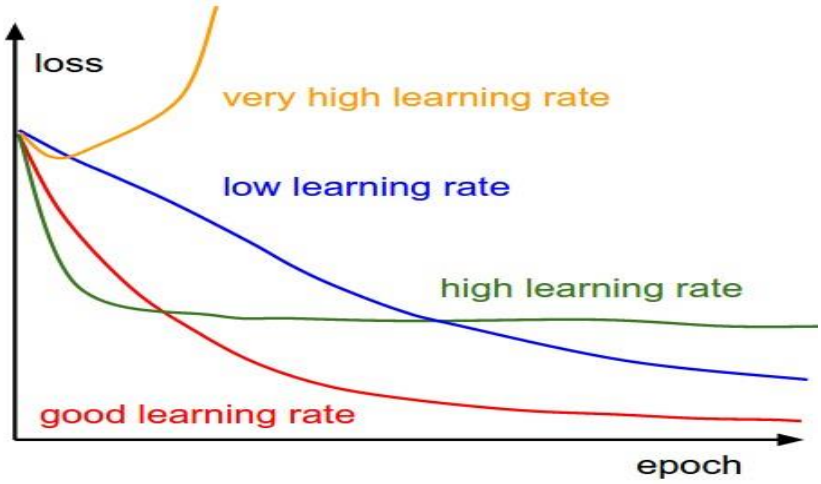
4.5 Model Evaluation and Optimization Algorithm

The model performance is evaluated by *categorical cross-entropy* and optimized by *Adam* algorithm that is mentioned before in sections 1.6, 1.5 respectively.

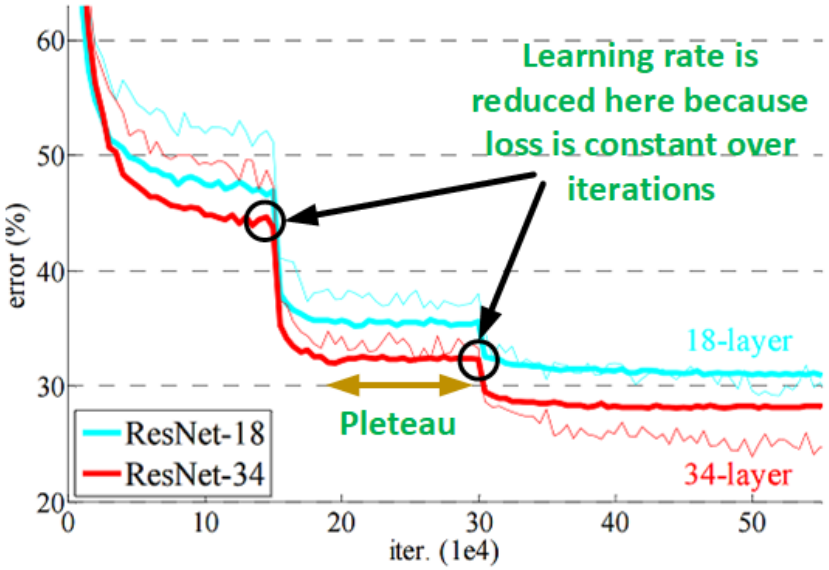
Although, we made some modifications to optimize our training process in this model.

It is known that the learning rate is one of the most critical hyper-parameters and has the potential to decide the fate of your deep learning algorithm. If you mess it up, then the optimizer might not be able to converge at all. It acts as a gate which controls how much the optimizer is updating the parameters w.r.t. gradient of loss.

The following figure explains the effects of learning rate on gradient descent. A very small learning rate will make gradient descent take small steps even if the gradient is big, thus slowing the process of learning. If the learning rate is high, then it becomes impossible to learn very small changes in the parameters needed to fine tune the model towards the end of the training process, so the error flattens out very early. If the learning rate is very high, then gradient descent takes big steps and jumps around. This can lead to divergence and thus increase the error.



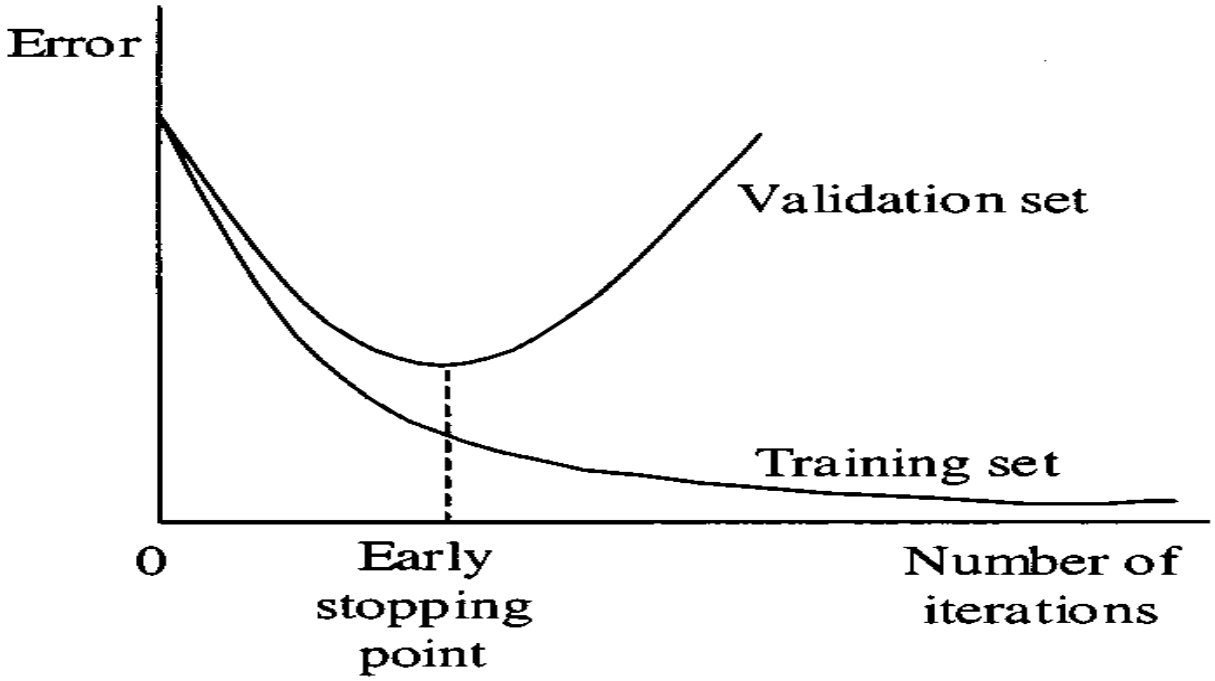
So, we first help the loss function to get rid of the "plateaus" by reducing the learning rate parameter of the optimization function with a certain value (factor) if there is no improvement on the value of the loss function for the validation set after a certain epoch (patience) as shown in the following figure:



Even if we could prevent the loss function from going to the plateaus, the value of the loss function of validation set could get stuck in a certain range while the training set does not (in other words, while the model continues to learn something). As long as we continue to train the model after this point, the only thing the model could do is to memorize (over-fit) the training data - it could be said that there is no chance of getting rid of the local minima for the loss function without a miracle -.This is something that we will not want at all.

So, we stop the training of the model if there is no change in the value of the loss function on the validation set for a certain epoch (patience) that is known as *Early Stopping*.

The following figure gives a better intuition about that:



4.6 Results

We got an accuracy of about 67.4% which is good with the FER2013 dataset compared to other implementations taking into account that the performances of winners' models in the relevant Kaggle competition were as follows:

1-RBM (Yichuan Tang) — 71.162%

2-UNSUPERVISED (Yingbo Zhou & Chetan Ramaiah) — 69.267%

3-MAXIM MILAKOV (Maxim Milakov) — 68.821%

4-RADU+MARIUS+CRISTI (Radu Ionescu & Marius Popescu & Cristian Grozea) — 67.484%

The model learning curve are illustrated in figure (a,b)

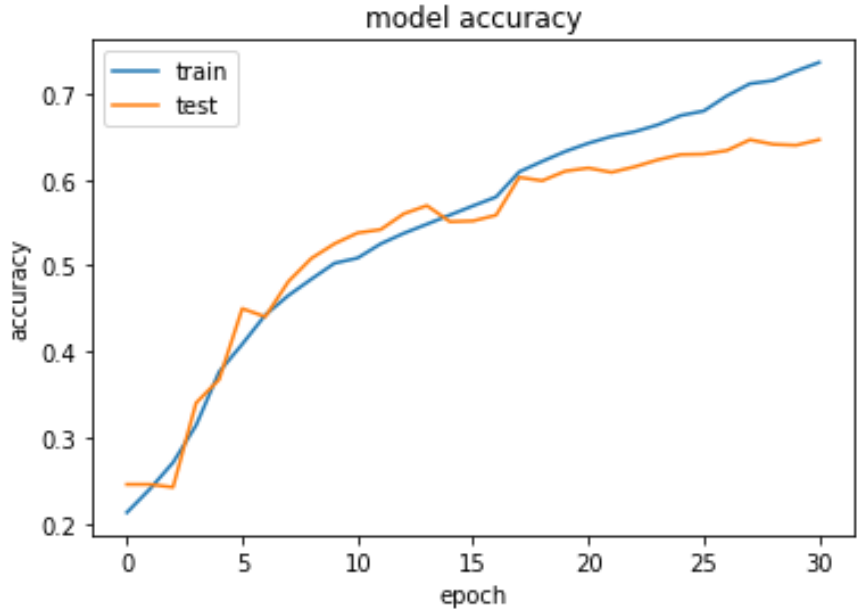


Figure (a)

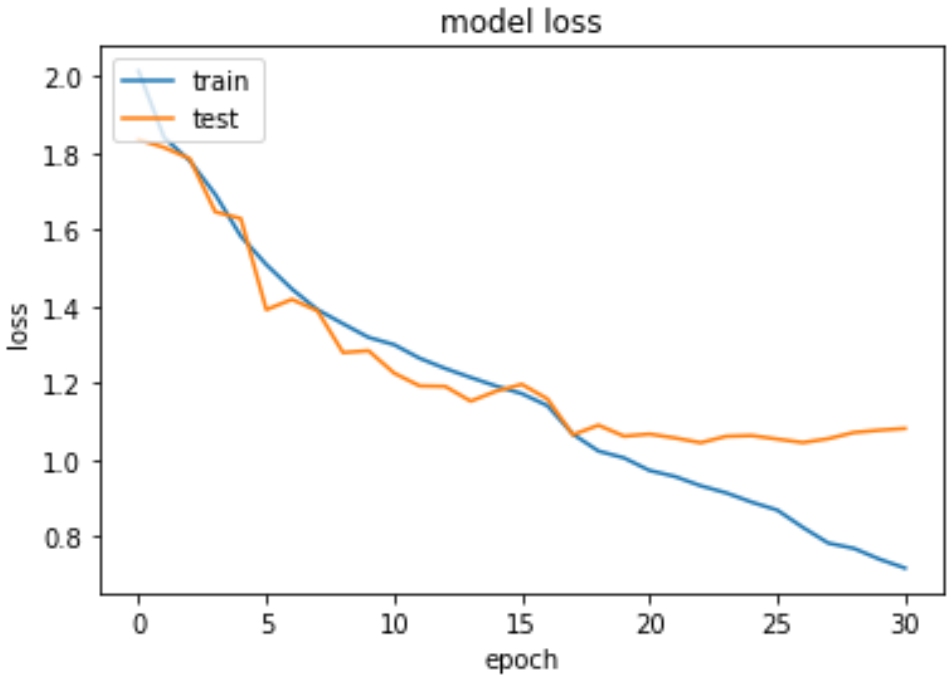


Figure (b)

Trying to test our model, it gives us better result than expected, it means that the FER2013 test dataset has some problems.

4.7 Future Work

The main challenge that we faced in this task was the lack of a good dataset, the images in FER2013 dataset have very small size which affects the performance and prevents us from using transfer learning techniques with a pre-trained state of the art model. So we hope in the near future to have an access to a suitable dataset to improve our performance and provide better assistance for visually impaired people.

4.8 References

1- Kaggle: Facial Expression Recognition Challenge.
2- Mishra S., Prasada G.R.B., Kumar R.K., Sanyal G. (2017) Emotion Recognition Through Facial Gestures - A Deep Learning Approach. In: Ghosh A., Pal R., Prasath R. (eds) Mining Intelligence and Knowledge Exploration. MIKE 2017. Lecture Notes in Computer Science, vol 10682. Springer, Cham.
3- Happy, S.L., George, A., Routray, A.: A real time facial expression classification system using Local Binary Patterns. In: Intelligent Human Computer Interaction (IHCI), 4th International Conference, pp. 1–5. IEEE (2012).
4- Lee, C.C., Mower, E., Busso, C., Lee, S., Narayanan, S.: Emotion recognition using a hierarchical binary decision tree approach. Speech Commun. 53(9), 1162–1171 (2011).

5- Lopes, A.T., de Aguiar, E., De Souza, A.F., Oliveira-Santos, T.: Facial expression recognition with Convolutional Neural Networks: coping with few data and the training sample order. Pattern Recogn. 61, 610–628 (2017).

6- Hu, T., De Silva, L.C., Sengupta, K.: A hybrid approach of NN and HMM for facial emotion classification. Pattern Recogn. Lett. 23(11), 1303–1310 (2002).

7- Sebe, N., Cohen, I., Gevers, T., Huang, T.S.: Emotion recognition based on joint visual and audio cues. In: 18th International Conference on Pattern Recognition, ICPR, vol. 1, pp. 1136–1139. IEEE, August 2006.

8- Liu, M., Wang, R., Li, S., Shan, S., Huang, Z., Chen, X.: Combining multiple kernel methods on riemannian manifold for emotion recognition in the wild. In: Proceedings of the 16th ACM International Conference on Multimodal Interaction, pp. 494–501 (2014).

9- The Subtle Art of Fixing and Modifying Learning Rate.

10- [Deep Learning Lab] Episode-3: fer2013.

5. Application Programming Interface

5.1 Abstract

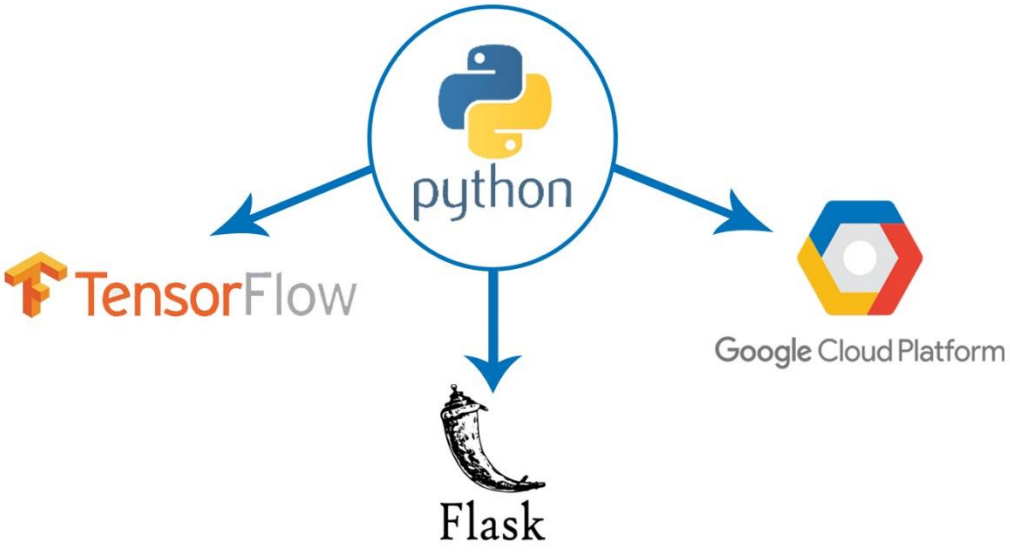
After building all the models separately, we need to define a way to use these models on any image, how it's sent and processed, and a specific way of how the result is returned. Here comes the role of building an API.

The API is written in Python using Flask web framework. It is more suitable than other frameworks for this application because it's lightweight (keeps the core simple but extensible) and Python supports the libraries that can process the trained models' data easier especially TensorFlow which deep learning is dependent on it.

The API contains 5 resources for the following services:

- Currency detection
- Image captioning
- Visual question answering
- Emotion detection
- Optical character recognition

Each one of these services' operation will be covered in the following sections as well as the helper functions that are used generally for all the applications.



5.2 Base64 Encoding

When sending images across the internet, the regular representation of an RGB array takes a lot of time and is prone to error. As an alternative, an encoding to images with base64 is made so that it can be received more reliably, so the API is ready to receive a base64 encoded image, then decode it and make the required processing.

Base64 is a byte encoder, whatever is passed to it, a picture, an mp3, or a string it takes its bit representation, separates it into a group of 6 bits, calculate the decimal representation of each group, then convert the decimal characters using base64 chart.

After the API processes the image and produces a text result, it converts the text result to sound bytes by a text-to-speech function (section 5.2), encodes these bytes with base64, and returns a JSON object with a "text" element containing a machine-text result, and a "sound" element containing base64 encoded sound bytes of the spoken text result.

Char	Value
A	0
B	1
C	2
D	3
E	4
F	5
G	6
H	7
I	8
J	9
K	10
L	11
M	12
N	13
O	14
P	15

Char	Value
Q	16
R	17
S	18
T	19
U	20
V	21
W	22
X	23
Y	24
Z	25
a	26
b	27
c	28
d	29
e	30
f	31

Char	Value
g	32
h	33
i	34
j	35
k	36
l	37
m	38
n	39
o	40
p	41
q	42
r	43
s	44
t	45
u	46
v	47

Char	Value
w	48
x	49
y	50
z	51
0	52
1	53
2	54
3	55
4	56
5	57
6	58
7	59
8	60
9	61
+	62
/	63

Base64 conversion chart

5.3 The Text-to-Speech Operation

Since the project's functionality can be used directly to help visually-impaired people, it's important that the result is not only produced in text form but is spoken in a hearable voice also.

Google Cloud Platform has a state-of-the art text-to-speech client with over 180 voices for different languages to choose from. They include pitch tuning so that the spoken text has a dynamic, real-life, feeling. An Arabic female voice is chosen for this API.

The process of building a *read_text* function that calls the Google Cloud client is as follows:

1- Creating a project on a GCP account, activating the text-to-speech API, and downloading the project credentials.

2- Adding an environmental variable in the python script with the location of the credentials.

3- Creating a synthesis object with the desired language and voice code ('ar' and female).

4- Taking the function's text argument and sending it with google.cloud.texttospeech python library.

5- Receiving an audio object containing byte-representation of the synthesized sound.

6- Encoding these sound bytes with base64 and returning it at the end of the function.

This service from Google Cloud Platform is charged at 4$/million characters. Another free text-to-speech library from google is gTTS with python but its voice is of lower quality.

A solution to reduce costs is possible from the observation that each of emotion detection, currency detection and visual question answering has a set of words or sentences that the result will fall into, so these options' sounds can be pre-saved in files with the API and read upon request. For image captioning, there a large

number of vocabularies which the model produces the result from so this vocabulary can be saved and processed the same way. Optical Character Recognition is the only application where this solution won't apply so to use it properly using GCP's text-to-speech API is a must.

5.4 Routes and Methods

For each one of five services included in this project, a route with the service's name is defined with a function that performs the service application.

The general route which is the link without any addition in the end e.g. http://127.0.0.1/ contains a simple message "Hello, welcome to IVA!" to just indicate that the API is working.

All of the routes take only a method of POST with a JSON object of the following form:

{

"image": *A base64 encoded image containing the wanted text*

}

Except the visual question answering which takes an extra argument "question" containing the text of the question on the image.

All the routes return a result as a JSON object of the following form:

{

"text": *the result in text form",

"sound": *the base64 encoded sound of the spoken text result*

}

Below is a brief description of the function of each route.

5.4.1 Currency Detection API

The currency detection model is discussed before in section (5.5). In the API, the currency detection function is reached by the following route:
http://127.0.0.1/currency

After an image is received, it is decoded from base64 to bytes and sent as an argument to Currency_Predict function which reads the *currency_model.h5* file with TensorFlow and returns one of the results for Egyptian currency from 5 pounds to 200 pounds.

If the maximum result score is less than 0.45 the text result is as follows:

"من فضلك أعد تصوير العملة"

If not, a text "جنيهات" is added to results 5 and 10, and "جنيها" is added to the rest of the currency to produce a viable sentence.

5.4.2 Emotion Detection API

The emotion detection model is discussed before in section (5.5). In the API, the emotion detection function is reached by the following route:
http://127.0.0.1/emotion

After an image is received, it is decoded from base64 to bytes and processed with two files:

- *haarcascade_frontalface_default.xml* - for face detection
- *gpu_mini_XCEPTION.63-0.64.hdf5* - the emotion detection model

A face detector and an emotion classifier are built from these files, then the image is resized to 400x400 and a result is obtained from the following options:

{ طبيعي، متفاجئ، حزين، سعيد، خائف، مشمئز، غاضب }

If no face is found then the result is the following text:

"عفوا لا نستطيع اكتشاف وجوه، حاول مرة أخرى"

5.4.3 Image Captioning API

The image captioning model is discussed before in section (5.5). In the API, the image captioning function is reached by the following route:
http://127.0.0.1/caption

After an image is received, it is decoded from base64 to bytes and processed with the following files:

- *word_index.pkl*
- *index_word.pkl*
- *embedding_matrix.pkl*
- *caption_model_weights.h5*

A result is produced with a sentence starter "يبدو كأنه" to indicate uncertainty, the text and sound results are then returned from the function.

5.4.4 Optical Character Recognition

Optical Character Recognition is the conversion of images of typed, handwritten or printed text into machine-encoded text using deep learning for processing.

Since it is a very important and much needed service, there has been a lot of research and pre-trained models to perform OCR on multiple languages. For our project, a pre-trained model from Google Cloud Platform services is used with this API. GCP's model is the best in the world right now for extracting text from hard images for Arabic, so it's the most viable option to help visually impaired people with reading from books with minimal error.

In the API, the OCR model can be reached from the following route:
http://127.0.0.1/ocr

And the mechanism of the OCR code calling GCP's API is as follows:

1- Creating a project on a GCP account, activating the vision API, and downloading the project credentials.

2- Adding an environmental variable in the python script with the location of the credentials.

3- Decoding the base64 string into a byte array and sending it with google.vision python library with their Image Annotator Client.

4- Receiving a container object and extract the text description.

5- Passing the text through the helper read_text function and convert the sound with base64 encoding.

6- Returning the text and sound as a JSON object with two elements, “text” containing the raw text response, and “sound” containing the base64 speech result.

This service from Google Cloud Platform is charged at 1.5$ per 1000 requests, when embedded with the application, it would require a subscription with an appropriate increment in the monthly cost.

5.4.5 Visual Question Answering API

The visual question answering model is discussed before in section (5.5). In the API, the VQA function can be reached from the following route:
http://127.0.0.1/vqa

The mechanism of the function is as follows:

1- The received image is decoded and passed through a tensorflow.keras Xception image model to extract the image features the same way the model was trained.

2- The question is passed through a *clean_str* function (same function that was used when pre-processing training questions see section (5.5).

3- The question is converted with *tokenizer.json* file, the saved tokenizer that was used on the training data to produce a number representation of the question words.

4- The image features and the tokenized question are given as an input to *vqa_xception_model.h5* file to produce answers.

5- The index with the maximum probability is extracted and the answer is obtained from *topAnsIndexWord.pkl* file which contains a mapping of each index to its corresponding text answer.

6- A text of "بنسبة" is concatenated with the answer then the probability of this answer is concatenated to the string to produce a viable sentence.

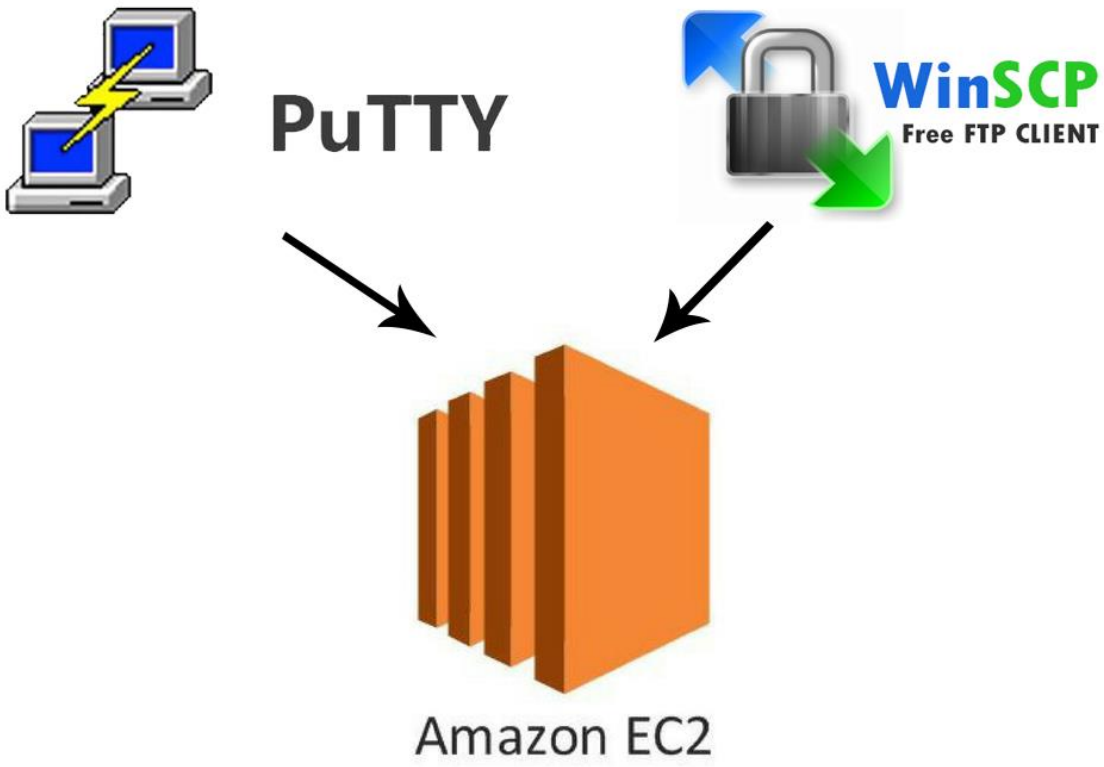
7- The text and sound results are then returned from the function.

5.5 Uploading the API to an Online Server (AWS)

After building a local python web app, the next step is to make it reachable from anywhere on the internet. There are multiple web app hosts that have been tried for this project, Heroku, AWS Elastic Beanstalk, but these hosts often have problems with installing python libraries for deep learning like TensorFlow and OpenCV. Thus, the best option was to build a manual web server on a virtual machine. Amazon Web Services is widely known for its strong infrastructure cloud computing services, therefore, using EC2 or Elastic Compute Cloud from AWS has turned out to be more flexible when it comes to installing libraries and defining timeout for requests.

Below is a description of each step in building the web server.

5.5.1 Amazon EC2



Amazon Elastic Compute Cloud (EC2) is a part of Amazon.com's cloud-computing platform, Amazon Web Services (AWS), that allows users to rent virtual computers on which to run their own computer applications. EC2 encourages scalable deployment of applications by providing a web service through which a user can boot an Amazon Machine Image (AMI) to configure a virtual machine, which Amazon calls an "instance", containing any software desired. A user can create, launch, and terminate server-instances as needed, paying by the second for active servers – hence the term "elastic". EC2 provides users with control over the geographical location of instances that allows for latency optimization and high levels of redundancy.

For this project, an instance with an Ubuntu 18.04 AMI, 4 GB RAM and 15 SSD capacity is created. The instance is created with a private SSH key pair so that it can only be accessed from an SSH client by the owner of the key only.

The security group for the instance allows access to all HTTP and HTTPS requests from any IP, and the owner of the private key can also access the AMI from any IP.

In order to access the instance Ubuntu terminal, PuTTY SSH client is used by converting the instance's private key pair extension to PuTTY's *.ppk* extension, and adding the instances public DNS address and port.

To transfer files to the Ubuntu instance, WinSCP software is used the same way as PuTTY, by adding the instance's public DNS and using the same private key pair.

WinSCP is a GUI-based file manager for Windows that allows uploading and transferring files to a remote computer using the SFTP, SCP, FTP, and FTPS protocols. WinSCP allows dragging and dropping files from a Windows computer to the Ubuntu instance or synchronize entire directory structures between the two systems. So, it makes transferring the whole directory of the local web app very easy.

5.5.2 NGINX

NGINX is open source software for web serving, reverse proxying, caching, load balancing, media streaming, and more.

When setting it up on the EC2 instance, it listens to a specified port (port 80 for this project) and any call to the instance public DNS to the process which runs on this port.

NGINX configuration can be tuned to make the optimal performance with the system its running on, an example of these configuration is:

- *worker_processes* – In most cases, running one worker process per CPU core works well.
- *worker_connections* – The maximum number of connections that each worker process can handle simultaneously. The default is 512, but most systems have enough resources to support a larger number. The appropriate setting depends on the size of the server and the nature of the traffic, and can be discovered through testing.

- *keepalive_requests* – The number of requests a client can make over a single keepalive connection. The default is 100, but a much higher value can be useful for testing.
- *keepalive_timeout* – How long an idle keepalive connection remains open.
- *keepalive* – The number of idle keepalive connections to an upstream server that remain open for each worker process.

As well as other variables that can tune access logging, limits, caching, but these are the most relevant to this project since its directed towards the server being called by a mobile application and are relevant when the application is being used by a lot of clients.

NGINX is the web server part of the project, it accepts requests, takes care of general domain logic and takes care of handling https connections. Only requests which are meant to arrive at the application are passed on toward the application server (Gunicorn) and the application itself (Flask web app).

5.5.3 Gunicorn

Gunicorn "Green Unicorn" is a Python Web Server Gateway Interface (WSGI) HTTP server. It is a way to make sure that the web server and python web application can talk to each other.

It takes care of everything which happens in-between the web server and the web application like:

- communicating with multiple web servers.
- reacting to lots of web requests at once and distributing the load.
- keeping multiple processes of the web application running.

Since sending images across the internet can take a long time even after encoding them with base64, Gunicorn has an advantage of configuring the timeout for the application so that it can be as large as needed. For this project, the request timeout is 360 seconds.

When uploading the Flask web application on the EC2 instance, it is configured to run on port 80. Gunicorn starts a process that points to this application with its configuration and NGINX automatically listens to this port and directs all

HTTP/HTTPS requests to the instance's public DNS to the application. In order to keep the Gunicorn process running after closing the PuTTY SSH connection, a process control system "Supervisor" is installed on the instance and used to monitor the process and keep access and error log files even when the connection is closed.

5.6 Future Work

The advantage of building a general-purpose application like Intelligent Vision Assistant, is that there is big room for improvement and additions. As an example, an extra service like object detection with 80 classes built from an open-source python library can be added to the API, as well as building a function specifically for color detection.

An additional service, face recognition, is already built with access to a NoSQL cloud database (MongoDB Atlas) to save known faces, but it needs to have a username and password protected application for privacy purposes.

As a future work in progress, text input like the question argument in VQA API can be replaced with sound bytes of the sentence and then converted with speech synthesis to obtain the text value of the question for a better experience to visually-impaired people. The same trick can be used with future application like face recognition when adding new faces.

When scaling the API to handle a large number of clients, multiple instances on EC2 would have to be created in addition to configuring a load balancer on AWS as well as using their CloudWatch service. Testing would be required at each stage based on the number of clients sending requests to the API, the statistics of these requests, and the capability of the virtual machines used to handle the requests.

5.7 References

- Google Cloud Text-to-Speech
- Google Cloud Vision
- AWS Elastic Compute Cloud
- Tuning Nginx
- Connecting to EC2 with PUTTY and WinSCP
- Gunicorn Documentation

6. IVA Android Mobile Application

6.1 Abstract

IVA is an android mobile application that targets individuals with visual impairments whose native language is Arabic in order to help them interact with their surroundings in an easier affordable way.

The mobile application is written in Java and Xml languages using Android Studio the official integrated development environment for Google's Android Operating System, ML kit library, Jetpack suite of libraries and others that will be mentioned.

IVA has a minimum SDK (software development kit) of 21, which means that it operates on android devices with android versions starting from 5.0 (Lollipop) and higher, that is more than 94% of the devices.

As the target audience of the mobile application is the visually impaired, there were important aspects that had to be carefully taken into consideration, such as Accessibility, screen readers and ease of use. All of them will be discussed further in later sections.

Android Studio

Java language

xml

Android Jetpack

6.2 Overall operation

IVA has a button for each service. When a user clicks a button, an image is taken then converted to a bitmap. The bitmap is then scaled, compressed, turned back into a byte array and encoded to a base64 string. By using JSON request and Volley Android libraries, the image is sent to the RESTful API that was previously discussed. A JSON response is now sent back to the user with a voice data encoded in base64 string and text.

Each of the steps and tools will be discussed in the following sections.

6.3 Layout and User Interface

To build a responsive User Interface Constraint Layout is used. It is a part of Android Jetpack suite of libraries. Constraint Layout is more flexible to use as views is placed relative to each other, it allows the designer to reduce the number of nested views hence, a better performance for layout files is achieved. In addition, it was designed to be simple as possible to avoid any confusion.

Figure 6.1 shows how the main activity of the mobile application looks, it is the first thing the user interacts with when he starts using the mobile application.

The button at the upper right corner is the flash button, with each click it changes the camera flashlight state of the device that are automatic, inactive and active states. Also, the state of the flashlight is audibly described in Arabic.

The four image buttons at the bottom of the figure resemble the four provided services which are the following from left to right:

- Optical character recognition.
- Emotion detection.
- Image captioning.
- Currency detection.

Figure 6.1

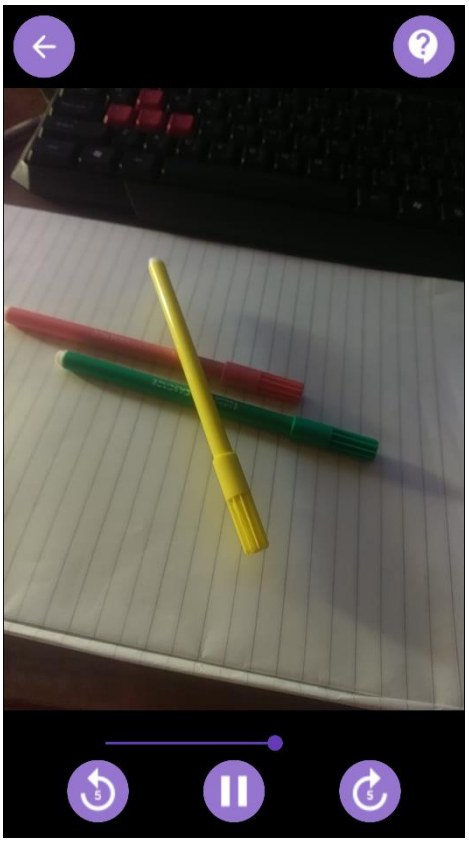


Figure 6.2

When each of them is clicked a short click, it takes an image and performs the service it represents. On the other hand, when any of these buttons, including the flashlight button, is long clicked a short description is played as an audio.

The most important element in figure 6.1 is the preview view, which is also a part of the Android Jetpack. As it's the view responsible for displaying the camera preview. The reason behind choosing the preview view is that it is suitable to use with the CameraX as it can be cropped, scaled and rotated for proper display.

The image preview streams to a surface inside the preview view when the camera becomes active.

Figure 6.2 is what the user interacts with after capturing an image as it shows the image and plays the result audio. Also, the bottom set of buttons and seek bar is responsible for playing, pausing, forwarding and replaying the audible media.

At the top right corner, a button is used to perform visual question answering, when clicked an alert dialog pops up to get the question from the user to send it to the API along with the encoded image.

Also, the same as figure 6.1 buttons, when any button is long clicked a description is played.

6.4 Camera

The application camera is built using CameraX API, it is a jetpack support library. Although it leverages the capabilities of camera2 API, it's a lot easier to use than camera2. In the beginning, IVA was built using camera2 API, it was very complicated as it needed a lot of set up. Also adding flashlight and zoom resulted in application crashes.

To solve those issues CameraX is used. It is life cycle aware while camera2 is not. As well as resolving device compatibility issues on its own so there is no need to include device-specific code in the code base.

Briefly, CameraX reduces the amount of code that needs to be written while adding the capabilities. In addition to easily managing consistent camera behavior such as aspect ratio, orientation, rotation, preview size and high-resolution image size.

6.4.1 Preview

CameraX introduces use cases, which allow you to focus on the task you need to get done instead of spending time managing device-specific nuances.

One of these use cases is the preview, it is used to display the scenery to be captured.

The steps followed to create a preview are:

1. Request a camera provider instance which is a singleton that can be used to bind the lifecycle of cameras to any lifecycle owner within the application process. Note that the life cycle owner is a class that has an Android lifecycle. Events can be used by custom components to handle lifecycle changes without implementing any code inside the Activity or the Fragment.
2. Create a preview use case builder instance.
3. Choose the desired camera whether lens facing front or lens facing back.
4. Bind the selected camera and preview use case builder to the life cycle.
5. Connect the preview view to the preview builder instance.

As zooming is also achieved in the preview use case it is done with the following steps:

1. Create a camera info interface that represents the information of the selected camera to get the current zoom ratio value.
2. Initialize a scale gesture detector with a scale gesture listener.
3. Whenever the preview view is pinched the gesture event is then sent to the listener.
4. Knowing the old zoom ratio value and the pinching area change a new zoom ratio is calculated.
5. By using the camera control interface, the new zoom ratio is assigned to the selected camera.

6.4.2 Image Capture

The second use case of CameraX is image capture. The image capture use case is designed for capturing high-resolution, high-quality photos and provides auto-white-balance, auto-exposure, and auto-focus functionality, in addition to simple manual camera controls.

An image capture builder is initialized with the desired functionality as mentioned above. Then, it binds to life cycle owner along with camera selector and preview use case builder.

The flash button in figure 6.1 is related to the image capture use case, as whenever the flash light state changes the image capture builder sets the flash light mode of the image capture builder as what is desired.

An important thing that needed to be taken into consideration is the camera sensor orientation and the rotation of image to be captured. Targeted rotation of the image is determined by the image capture builder. As a result, orientation event listener is used. Whenever the mobile phone orientation changes the image capture builder target rotation changes to match up with it.

The reader might question the importance of keeping track of the orientation. For example, if an image of a face is captured to detect emotion it has to be posted to the API in a way the face is shown in figure 6.3 to be able to receive a response, if it were to be sent in the way shown in figure 6.4 that would cause a problem of not detecting a face. Same thing goes for optical character recognition, image captioning and visual question answering.

Which means that however the user holds the mobile phone, the image will be captured with the proper rotation.

Figure 6.3

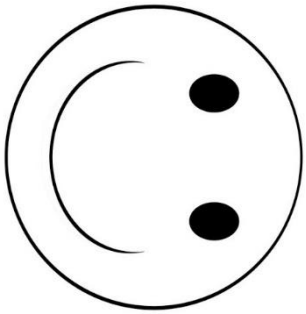


Figure 6.4

6.4.3 Image Analysis

The image analysis is the third use case that provides the application with a CPU-accessible image to perform image processing, computer vision, or machine learning inference on. The application implements an analyze method that is run on each frame.

An image analysis builder is initialized with the chosen functionality and bind to life cycle owner. So now we have the three use cases and camera selector bound to the life cycle owner.

Images are processed by passing an executor in which the image analysis is run and an instance of a class that implements *ImageAnalysis.analyzer* as parameters to the set analyzer method.

6.4.4 ML Kit and Face Detection

To ease the use of the application ML kit is used, ML Kit brings Google's machine learning expertise to mobile developers in a powerful and easy-to-use package. As mentioned in the previous section, image analysis is used to analyze the camera frames with the help of ML kit to detect faces in order to help the user take images of faces in a better way.

With ML Kit's face detection API, faces can be detected in an image, identify key facial features, and get the contours of detected faces.

Face detection API has key capabilities such as:

- Recognize and locate facial features. Get the coordinates of the eyes, ears, cheeks, nose, and mouth of every face detected.
- Get the contours of facial features. Get the contours of detected faces and their eyes, eyebrows, lips, and nose.
- Recognize facial expressions. Determine whether a person is smiling or has their eyes closed.
- Process frames in real time. Face detection is performed on the device, and is fast enough to be used in real-time applications.

Contouring the detected face is indeed not very helpful for the visually impaired. So, whenever a face is detected an indication sound is played to notify the user that there is a face detected in the camera preview. Detected face is then tracked; figure 6.5 shows a detected face surrounded by a rectangle.

Figure 6.5

6.4.5 Image Taking

After initializing all the use cases and binding them to the life cycle owner, the camera is ready to be used and capture images. The image capture builder is used to do so with the take picture method. An executer and an on image captured callback are the parameters of the method.

When any of the four buttons at the bottom of figure 6.1 is clicked, an image is taken. After that, the image goes through multiple steps before posting it to the API.

If the image is taken successfully, then these steps are followed to get the base64 encoded image (base64 was discussed in section 5.2):

1. Image rotation is stored as an integer value, that is because on some devices the desired performance of orientation event listener is not achieved.
2. The image is then converted to a byte array.
3. The byte array is decoded to a bitmap.
4. After getting the bitmap, the rotation is checked to be modified to the desired one in order to get a rotated bitmap.
5. The rotated bitmap is then displayed in an image view.
6. After that a scaled bitmap is generated.
7. Scaled bitmap is then compressed to JPEG format and then converted to byte stream.
8. Byte stream is converted to a byte array.
9. The last step is that the byte array is now encoded to a base64 string.

Compressing the bitmap is for reducing the size of the output byte stream while maintaining quality, which is very necessary to speed up the post request.

6.5 Networking and Volley library

The step after encoding the captured image is posting it to the API to process it. The JSON object sent has the form that was mentioned in section 5.4 which is the following:

{"image": "the result of encoding the image to base64 string"}

In the case of visual question answering it has the following form:

{"image": "the result of encoding the image to base64 string",

"question": "String"}

For posting the JSON object to the API Volley library is used, it is an HTTP library that makes networking for Android apps easier and most importantly, faster. Volley offers several benefits such as:

- Automatic scheduling for network requests.
- Cancellation of the request.
- Ease of customization, for example, for retry and back off.

- Strong ordering that makes it easy to correctly populate your UI with data fetched asynchronously from the network.
- Transparent disk and memory response caching with standard HTTP cache coherence**.**
- Multiple concurrent network connections.
- Support for request prioritization.
- Integration with any protocol and comes out of the box with support for raw strings, images, and JSON.

The JSON object is posted to the API using a standard JSON volley request, which is a subclass of the JSON request. The response of the request is also a JSON object which has the following form:

{"sound": "base64 encoded sound string",

"text": "the same as sound but written as string"}

There are four important things to post the request as shown in figure 6.6:

- Chose the method of the request to be post.
- Pass the URL the JSON object is going to be posted to.
- The created JSON object with the image encoding string.
- A response listener to check if the request was a success or if there were any error.

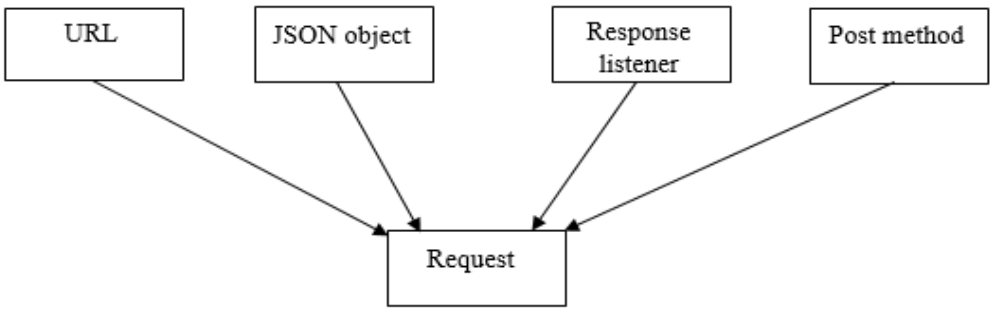


Figure 6.6

If the request is a success, then the received JSON object is processed so that the user will be able to get a result for one of the services. If it's not, the response will be a volley error defining what type of network error has occurred. The types of volley network errors are the following:

- **Network error** is due to bad internet connectivity or other network errors.
- **Server error** can happen due to several reasons, one of them if there's a null value passed to the request in this case the request is bad. Another reason is if the URL used is bad or has a typo. Also, if the JSON object keys don't match up with the keys the API is expecting.
- **Authentication failure error** would happen if there's a problem connecting to the server. This error shouldn't be occurring at the current time because IVA doesn't have any authentication parameters passed to the request.
- **Parsing error** is as a result of receiving different values than expected and using get method instead of post.
- **No connection error** happens when the device is not connected to the internet whether with WIFI or cellular data.
- **Timeout error** is when the request takes long time without getting a response. Although volley library has a retry policy, a maximum number or retries is set and timeout error may occur.

If any of these errors occur while posting the JSON object to the API, an alert dialog is shown and audible feedback is played to inform the user about what kind of error has occurred and ask whether he wants to retry or not.

Another thing that is need to be taken care of is the network security configuration. The Network Security Configuration feature lets apps customize their network security settings in a safe, declarative configuration file without modifying app code. These settings can be configured for specific domains and for a specific app. The key capabilities of this feature are as follows:

- **Custom trust anchors:** Customize which Certificate Authorities (CA) are trusted for an app's secure connections. For example, trusting particular self-signed certificates or restricting the set of public CAs that the app trusts.

- **Debug-only overrides:** Safely debug secure connections in an app without added risk to the installed base.

- **Clear text traffic opt-out:** Protect apps from accidental usage of clear text traffic.

- **Certificate pinning:** Restrict an app's secure connection to particular certificates.

The Network Security Configuration feature uses an XML file where the network settings are specified for the application. An entry must be included in the manifest of the application to point to this file.

Starting from android version 9 (pie), the network security configuration file should be added to the manifest to declare to the application that the API URL is a secure connection by configuring its certificate authority, as it is a Http protocol. While using protocols such as Https is trusted. Lower android versions trust certificate authorities by default.

6.6 Sound and Media Player

After the response is sent back to the user, the application retrieves the encoded sound from the JSON object and then decodes it to a byte array, that byte array is then converted into mp3 file to be played by Media Player which a class used to control the playback of audio files and streams.

Figure 6.7 shows the life cycle and the states of a Media Player object driven by the supported playback control operations. The ovals represent the states a Media Player object may reside in. The arcs represent the playback control operations that drive the object state transition. There are two types of arcs. The arcs with a single arrow head represent synchronous method calls, while those with a double arrow head represent asynchronous method calls.

Two or more Android apps can play audio to the same output stream simultaneously. The system mixes everything together. While this is technically

impressive, it can be very aggravating to a user resulting in bad user experience. So the solution for that is audio focus requesting.

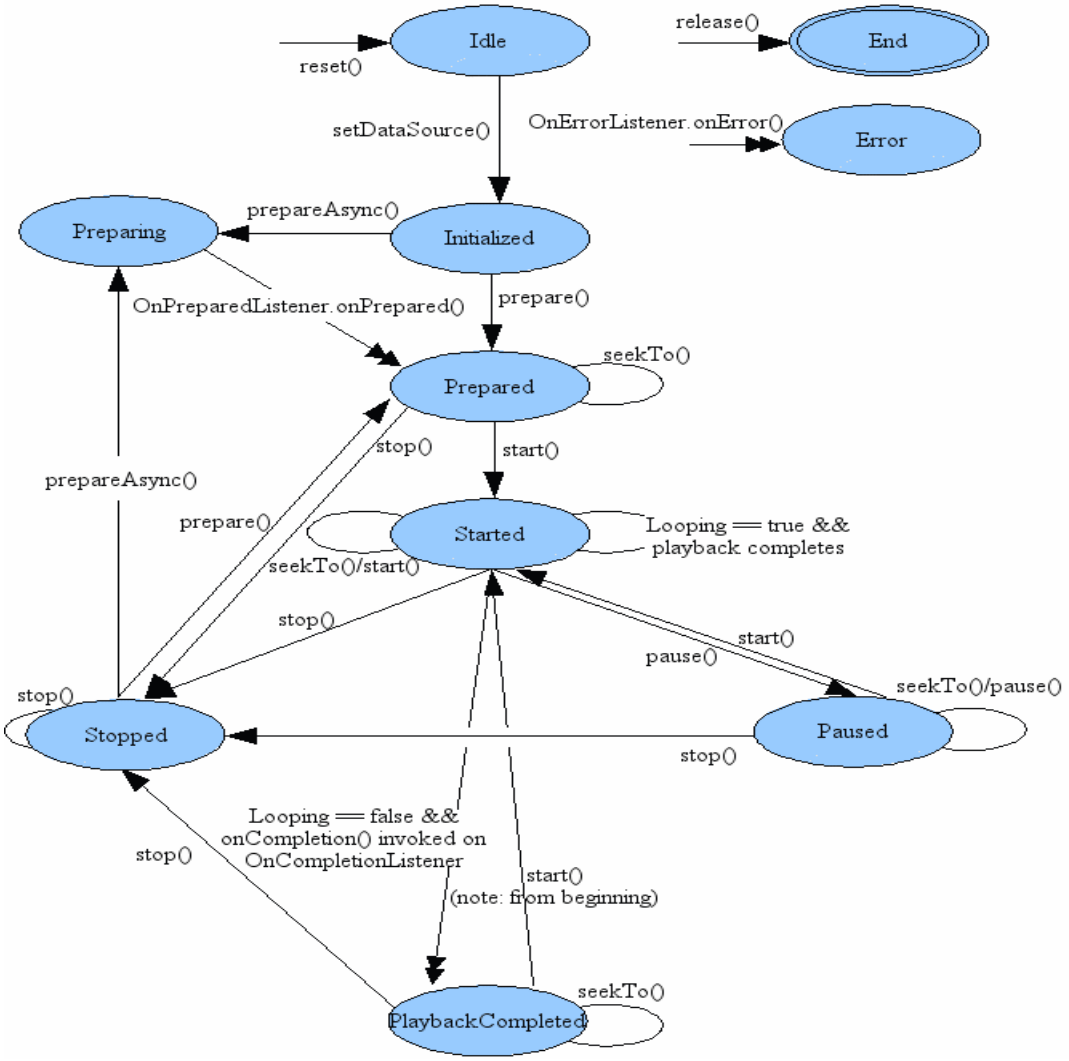


Figure 6.7

When a certain application wants to play audio, it should request audio focus, if the focus is granted, it can play the sound. However, another application can request audio focus at the same time if it gets it the first application has to pause playing or lower the volume. An important thing to do after the media player completes playing is to abandon audio focus if the application no longer needs it to allow other applications to grant it whether to start playing a sound or to continue a paused one.

Audio focus has multiple states that it can change to after the request is granted, those changes occur when another application requests for the audio focus while playing an audio, and passed to the audio focus change listener so the application can handle it, some of those changes are:

- **Transient loss of focus**: during the transient loss of the audio focus the application should continue monitor changes in audio focus and be prepared to resume normal playback when you regain the focus. When the other application abandons audio focus, the application should now gain the audio focus and be able to resume playing or restart.

- **Permanent loss of focus**: when the audio focus is lost permanently, the application should stop playing at once, and it won't be able to gain the audio focus when the other application abandons it. So audio will be played when the user takes an action.

Figure 6.2 shows a seek bar and three buttons, they are used for playing, replaying, and forwarding. The bar is used to show the progress of the audio playing, and also to seek to a certain point of the audio.

6.7 Accessibility and Talk Back

When any application is designed, it should be usable by all people including people with disabilities. As IVA mainly targets the visually impaired, then it's a must to make it more accessible for the best user experience.

6.7.1 Accessibility

In order to help people with accessibility needs use the application successfully, there are some practices that needed to be followed while designing IVA:

- Label elements: it's important to use descriptive labels for each interactive UI element in the application. Each label should explain the meaning and

purpose of a particular element. Screen readers such as Talk Back can announce these labels to users who rely on these services which will be discussed in the next section.

- Increase the text visibility: by selecting the proper text size and contrast ratio between the text and back ground.

- Use large simple controls: application buttons or any interactive element in the UI should have a minimum touch target of 48X48 dp (density-independent pixel), One dp is a virtual pixel unit that's roughly equal to one pixel on a medium-density screen (160dpi; the "baseline" density).

- Describe the UI element: each element should have a description identified in the view XML content description attribute; screen readers also rely on this description.

- Make media content more accessible: by adding descriptions to the application audio content so that users who consume this content don't need to rely on entirely visual cues.

6.7.2 Talk Back screen reader

Helps people who have low vision or are blind. Announces content through a synthesized voice, and performs actions on an app in response to user gestures.

In figure 6.8 a green triangle surrounding the image captioning button can be seen, this triangle is a part of the talk back screen reader. The user can use swiping right and left and other gestures to navigate through the views on the screen. During navigation, the label or content description of the view is announced by synthesized voice. If the view is an interactive one, then the talk back tells the user what actions can be done, a double click for a short click and a double click and hold for long press.

The user selects the desired language and Text to Speech engine from the settings. The long press feature was added to IVA because in some mobile phones the

Arabic Text to Speech language is not available so the screen reader won't announce an Arabic description, so it's an alternative way to use the talk back to interact with the application and also get a description.

Figure 6.8