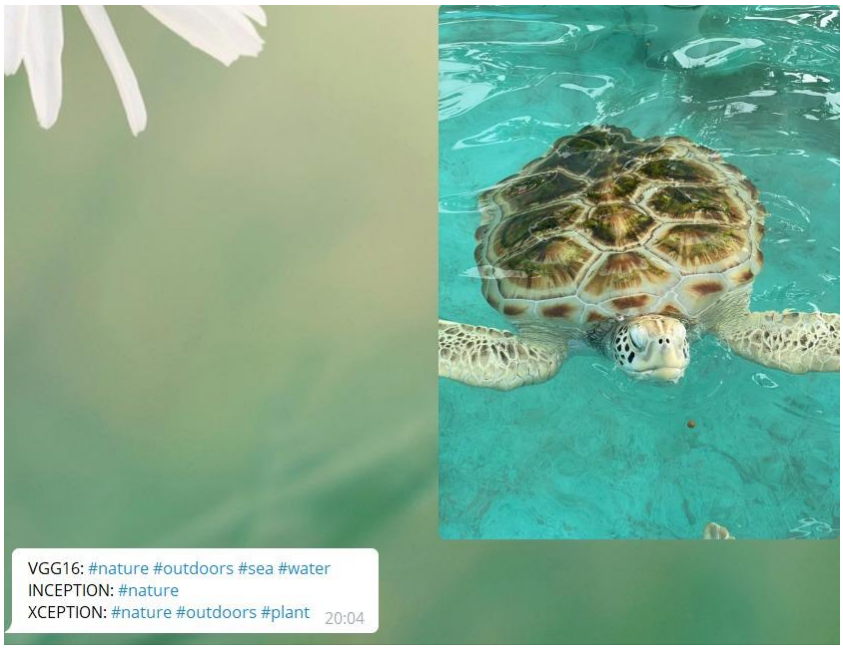
# Hashtag # - Image Classification Bot

**Project description:**

The project goal is creating a model that classifies images and labels them by common tags (multi label classification model). The classifier would be deployed on a telegram bot, allowing real time prediction for users.

The model trained on images from Unsplash dataset: https://unsplash.com/

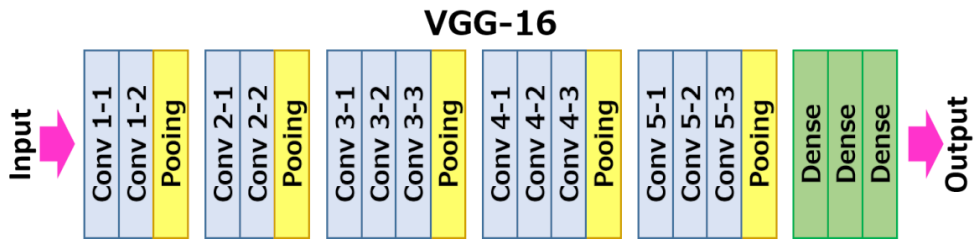



There are many types of models (or algorithms) that can help us achieve the project goal; among them are: ResNet, Inception, VGG and Xception.

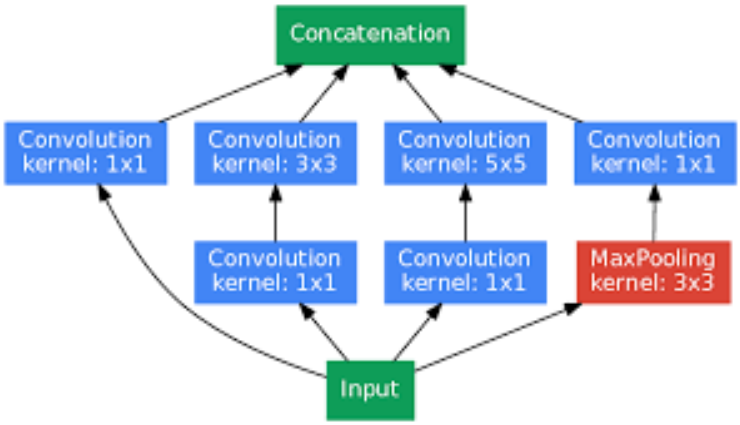
In our project we decided to focus on VGG16 , Inception V3 and XCeption and compare the results of the different models.

The reason we chose them is that they are pre-trained models and they come from different families of models (The architecture of VGG16 is considerably different from the architecture of Inception family) and we thought it is interesting to evaluate their parameters, accuracy and overall performance utilizing the same dataset.

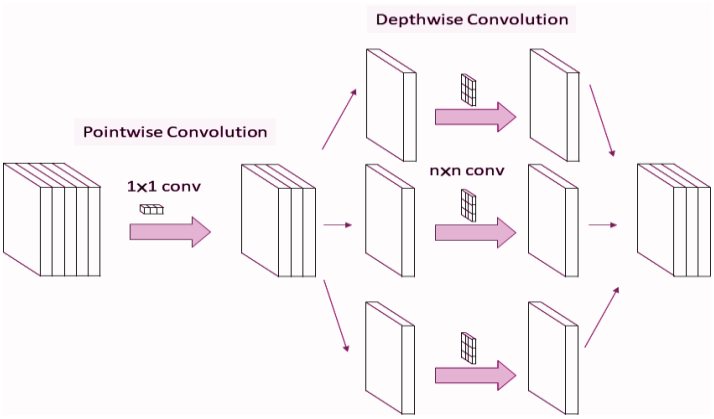
A schemes of the structure of these models:



**Inception V3:**




**XCeption:**




We considered VGG16 as our baseline model, however it was heavier and we felt we might be able to obtain similar results with a smaller cost utilizing Inception models.

Inception networks were the first to introduce multiple convolutions per layer, rather than a sequential convolution as was seen previously in VGG and RESNET. It also includes residual connections between layers, which are arbitrarily dropped and help reduce the total cost of the model. In original Inception net (GoogLeNet), there were depth wise convolution layers (with varying kernel sizes) followed by 1*1 convolution layer (pointwise convolution). Following Inception V3 release, the order was reversed and the pointwise convolution was initiated prior to the depth wise convolution. It helped reduce the total number of weights associated with each convolution as well as improved the accuracy by allowing a faster convergence. The XCeption net introduced another advancement - the intermediate ReLU nonlinearity was dropped, and so the complexity of the net was reduced, and convergence was achieved faster.

**Dataset + EDA:**

We downloaded the dataset from this link: https://github.com/unsplash/datasets

The dataset contains 4 tables with details about the images:

Conversion table:

```
[35] conversions_df = datasets['conversions']
     print(conversions_df.shape)
     conversions_df.head()
```

```
(4075504, 6)
```

| | converted_at | conversion_type | keyword | photo_id | anonymous_user_id | conversion_country |
|---|---|---|---|---|---|---|
| 0 | 2020-02-28 17:53:59 | download | ancient cave art | 5vV1xgPPdoI | 9dfbbbda-cf6c-4bf8-b519-9082d82e46ce | US |
| 1 | 2020-02-28 18:00:43 | download | bali | -zxG8fyqCg0 | 11a88646-226c-4ff0-a0e6-0ad1870f5422 | IN |
| 2 | 2020-02-28 18:08:42 | download | water | ebbFZvavGy4 | 38a51244-0dae-4ee2-b08b-446899879e3a | US |
| 3 | 2020-02-28 18:10:22 | download | jungle | nNNxBHA4ops | a85b3b56-9fcd-4846-a8ac-e232d9cfd912 | AR |
| 4 | 2020-02-28 18:14:33 | download | simple background | RAVdOqWXPvg | 4fc76af5-e8f7-459b-8b6a-a0539dc26e10 | HU |

Collections table:

```
[34] collections_df = datasets['collections'].head()
     print(collections_df.shape)
     collections_df
```

```
(5, 4)
```

| | photo_id | collection_id | collection_title | photo_collected_at |
|---|---|---|---|---|
| 0 | --2lBUMom1I | 4668070 | Pose | 2019-04-18 23:59:25 |
| 1 | --2lBUMom1I | 162470 | Majestical Sunsets | 2016-03-15 17:04:25 |
| 2 | --2lBUMom1I | 4472213 | People | 2019-03-17 04:44:02 |
| 3 | --2lBUMom1I | 2143051 | Travel / Places | 2018-05-22 23:20:05 |
| 4 | --2lBUMom1I | 3734077 | Images | 2018-12-30 00:56:45 |

Keywords table:

```
key_words_df = datasets['keywords']
print(key_words_df.shape)
key_words_df.head()
```

```
(2689739, 5)
```

| | photo_id | keyword | ai_service_1_confidence | ai_service_2_confidence | suggested_by_user |
|---|---|---|---|---|---|
| 0 | zzux2cH-F-A | hat | 34.696083 | NaN | f |
| 1 | zzux2cH-F-A | fog | 41.619389 | NaN | f |
| 2 | zzux2cH-F-A | arenaria | 40.862339 | NaN | f |
| 3 | zzux2cH-F-A | mist | 40.306744 | NaN | f |
| 4 | zzux2cH-F-A | ice | 60.944134 | NaN | f |

And photos table:

```
[47] photos_df = datasets['photos']
     print(photos_df.shape)
     photos_df.head()
```

```
(25000, 26)
```

| | photo_id | photo_url | photo_image_url | photo_submitted_at | photo_featured | photographer_username | photographer_first_name |
|---|---|---|---|---|---|---|---|
| 0 | 8ZgJyLGbC7Y | https://unsplash.com/photos/8ZgJyLGbC7Y | https://images.unsplash.com/40/KJyFV5SZSweiYGh... | 2014-07-05 13:43:36 | t | martindorsch | Martin |
| 1 | -_rYK0egLWE | https://unsplash.com/photos/-_rYK0egLWE | https://images.unsplash.com/31/xDtuvK3GRJGUSUH... | 2014-03-18 16:35:04 | t | oliviahenry | Olivia |
| 2 | PewUcrT1ylw | https://unsplash.com/photos/PewUcrT1ylw | https://images.unsplash.com/photo-141535311598... | 2014-11-07 09:44:28 | t | freephotosbydawn | Dawn |
| 3 | kFxWDfj0pD8 | https://unsplash.com/photos/kFxWDfj0pD8 | https://images.unsplash.com/photo-142224671965... | 2015-01-26 04:20:40 | t | alexjones | Alex |
| 4 | r3ZWnitp3zk | https://unsplash.com/photos/r3ZWnitp3zk | https://images.unsplash.com/photo-141582700792... | 2014-11-12 21:17:08 | t | envisual | Charlie |

Only the keywords and the photos tables were important for our case.

For our classification model we needed the URL of the image and the tag of the image.

We decided to focus on ten common tags, which are the most frequent in the dataset: animal, human, mountain, nature, outdoors, person, plant, sea, tree and water.

We then downloaded the numpy arrays of all the images with corresponding tags and saved them to a numpy array which was transformed to pickle files. In the process of downloading the arrays we resized them to 224*224 and changed the color format to be a uniform RGB. The keywords were filtered by confidence interval, and we only allowed tags with at least 99% certainty. The label dataframe was converted into a one hot encoded dataframe, containing 10 columns - 1 for each tag. It was also saved as a pickle file.

Except for the size of the images and filtering images by specific tags, we haven't changed the dataset.

**Implementation:**

1. VGG16: Using the default input size (224,224,3) and pretrained weights, we removed the top layer and froze the rest. We added three layers (flatten, dense-Relu(128), dense-sigmoid(10), and compiled the model with binary cross entropy loss function, and Adam optimizer. The data was split to 80% train and 20% test sets, and a 10% validation set taken from the train set.
   After 5 epochs, we achieved ~65% accuracy for this multilabel classifier.
2. Inception V3: Inception and the following model, XCeption, were originally trained on 299*299*3 images. We trained to obtain our images in these dimensions, however they were extremely large and even normalising the values between 0-1 did not improve the situation. We therefore decided to use the same 224*224*3 images that were used in the VGG16 model. We added three layers (flatten, dense-Relu(128), dense-sigmoid(10), and compiled the model with binary cross entropy loss function, and Adam optimizer. The data was split to 80% train and 20% test sets, and a 10% validation set taken from the train set.
   After 4 epochs, we obtained ~80% accuracy.
3. XCeption had a similar architecture to Inception V3, including the additional three layers.
   After 6 epochs, XCeption achieved ~76% accuracy.

   **Experimenting with architectures:**

   During the exploration of the models' architectures, we experimented with data augmentation which included horizontal flip and rotation of up to 90 degrees. Unfortunately, since the dataset was large and heavy, the augmentation was not

feasible and caused crashes due to low RAM memory. We tried to convert the models to work on TPU, however it was not successful and therefore we dropped the augmentation part.

We also experimented with different architectures for the additional 3 layers, namely using global average pooling or max pooling rather than using a flatten layer. They did not improve the model's accuracy but only worsened the prediction and were therefore deemed unsuccessful.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 128)               3211392
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 17,927,370
Trainable params: 17,927,370
Non-trainable params: 0
```

```
max_model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Model)         (None, 5, 5, 2048)        21802784
_________________________________________________________________
flatten (Flatten)            (None, 51200)             0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               6553728
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1290
=================================================================
Total params: 28,357,802
Trainable params: 6,555,018
Non-trainable params: 21,802,784
_________________________________________________________________
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
xception (Model)             (None, 7, 7, 2048)        20861480
_________________________________________________________________
flatten (Flatten)            (None, 100352)            0
_________________________________________________________________
dense (Dense)                (None, 128)               12845184
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 33,707,954
Trainable params: 33,653,426
Non-trainable params: 54,528
```

**Serving the model:**

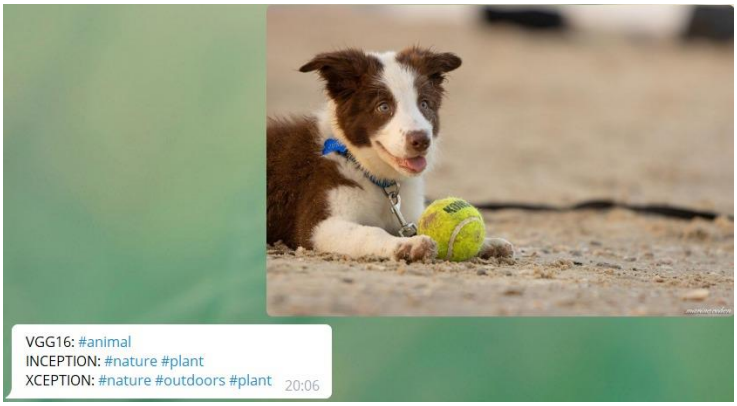
To serve our model we chose a special API – Telegram bot.

To build the bot, we used a library named: `python-telegram-bot`.

The bot algorithm loaded the weights of each model simultaneously – VGG16, XCeption and Inception V3, and had a predefined list of our tags.

Each image that is received by the bot, is converted to shape (224, 224, 3) and then each model suggests its prediction to the processed image. The output contains the keyword predictions of each model separately.

If you want to try the bot, run the telegram notebook from the drive folder and use the following link to ask for a prediction:




https://t.me/Pic_hashtag_bot


**Models evaluation:**

The models offer accuracy rates of 65-80%. It overall sounds fairly good for multilabel classifiers, however when checking the predicted keywords we see that VGG outperforms the inception models, and offers more accurate predictions, from more categories. We suspect that the inception models are more inclined to offer nature and outdoors tags as the dataset is imbalanced, and these two tags have greater representation in the train dataset.

As the inception model was trained on (299,299) pixel size and we forced a smaller size, we suspect it affected the model's performance. Our plan to reshape the data was deemed unsuccessful and so the Inception models require further training to compensate.

All three models could have benefitted from data augmentation but due to resources crisis it was not achieved.

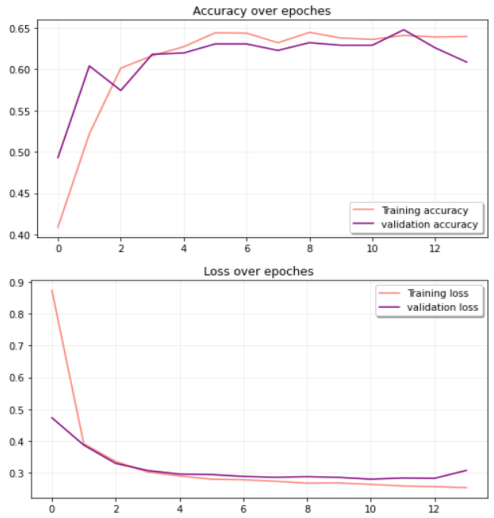
**Challenges:**

- Our main challenge was trying to work with Cnvrg's environment.
  The resources of Google Colab notebook are limited and we hoped to have better performances using Cnvrg's environment.
  Cnvrg's environment has many technical issues such as opening a workspace and running our code with inconsistent results, so we resorted to Google Colab.

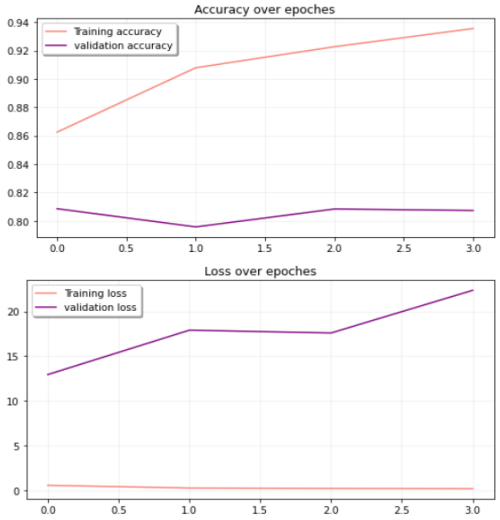
- The original dataset consisted of a list of links directing to the image web page. The images had large and varying dimensions (over 1024*1024) and when we downloaded them one by one it took a long time.
  To overcome this we saved the pixel arrays as numpy arrays which were saved as 13 pickled files.
- The current performance of the models serves as a great challenge as we are looking at different ways of improving it: changing learning rate, manipulating the base model layers or adding new layers, obtaining more images or utilising image augmentation.

The accuracy and loss values of the three models is presented below:

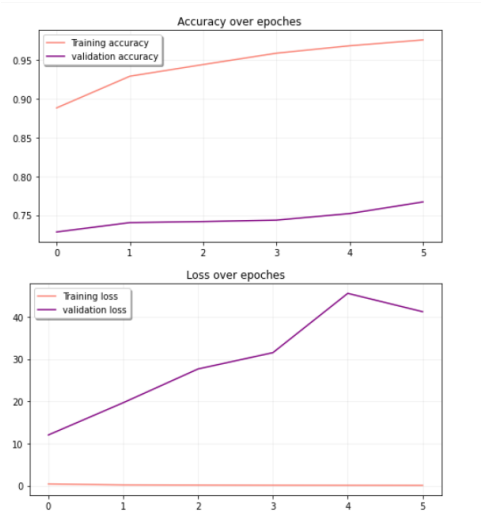
**VGG16:**




**Inception V3:**



**XCeption:**




**Directions for future work on the problem:**

1. Preparing the data took time, both because we were required to transfer a large amount of images from the website, and because each time we realized that preprocessing of the images in advance was needed to optimize the data. After resizing all images to the same dimension (because the original images came in different, huge resolutions), we saw that some networks were trained using different dimensions than what we obtained. On second thought - we would have made the sizes of the images according to the requirements of the network before preparing the data, assuming there will not be a space limitation.
2. If the technical limitations did not hinder us - we would have liked to see how the VGG16 model behaves when all the layers are trained and with a different learning rate. In addition, we would have tested the model on a larger dataset, with additional keywords, and see the different effects of data augmentation.
3. Test additional networks if the images were adjusted to their requirements.
4. Add a title or caption to an image, and not only the probability of matching keywords.
5. We had another idea that we could apply – scrap images and tags from Instagram and train our model based on them.