Meme Image Classification

THIS REPORT IS SUBMITTED TO **PROF. SHOBHA BAGAI, DR. NIRMAL YADAV AND DR. HARENDRA PAL SINGH** IN THE FULFILLMENT OF MONTH-LONG PROJECT IN THE FIELD OF NUMERICAL METHODS AT B. TECH. VTH SEMESTER, CIC

Aayush Jain | B. Tech. Vth Sem | 11601
Aditya Sharma | B. Tech. Vth Sem | 11602
Vaibhav Jain | B. Tech. Vth Sem | 11634

ABSTRACT

Internet is filled with memes. No matter which social platform, you’ll find a pepe, deal with it or bad luck brian on it. Memes have taken over the society. European Union has now passed a bill that can perish this culture. In an attempt to restrict piracy and enforce copyright laws, EU might make various software companies deploy filters that can help prevent upload of copyrighted content by an unauthorized individual or company. This will also end up stopping people from uploading memes on any such platform. This is because most memes are generated from templates and the filter won't be able to easily discriminate between such content. In this project we have built a classifier which can act as a base for an intelligent upload filter that can specifically distinguish among a variety of memes.

Introduction

What is a meme?

The word meme is a shortening of mimeme (from Ancient Greek) ,"imitated thing", coined by British evolutionary biologist Richard Dawkins in The Selfish Gene (1976)[1][2] as a concept for discussion of evolutionary principles in explaining the spread of ideas and cultural phenomena. The thought behind meme was peer to peer spreading of an idea, behaviour or style within a cult with a propaganda of conveying some theme or phenomenon carried along by the meme itself. A meme was supposed to act as a unit for carrying cultural ideas, symbols, or practices, that can be transmitted from one mind to another through writing, speech, gestures, rituals, or other imitable phenomena with a mimicked theme.

Supporters theorize that memes are a viral phenomenon that may evolve by natural selection through the processes of variation, mutation, competition, and inheritance in a manner analogous to that of biological evolution. Memes that propagate less prolifically may become extinct, while others may survive, spread, and (for better or for worse) mutate. Memes that replicate most effectively enjoy more success, and some may replicate effectively even when they prove to be unfavourable to the welfare of their hosts.

An Internet meme, a more common form of a meme is an activity, concept, catchphrase, or piece of media that spreads, often as mimicry or for humorous purposes, from person to person via the Internet. An Internet meme usually takes the form of an image, GIF or video. It may be just a word or phrase, sometimes including intentional misspellings (such as in lolcats). These small movements tend to spread from person to person via social networks, blogs, direct email, or news sources.

Motivation for this Project

Online content as such memes are replicas of the protected unauthorised word whose spreading violates the copyrights of its original owner. The European Parliament has voted in favour of a controversial new copyright directive that could force tech giants to do much more to stop the spread of such copyrighted material on their platforms. The European Union Directive on Copyright in the Digital Single Market, to give it its full name, is designed to update existing copyright laws for the internet age.

The Directive on Copyright is sometimes referred to as ‘Article 13’ after its most controversial component – the article that would require online platforms to filter or remove copyrighted material from their websites. It’s this article that people think could be interpreted as requiring platforms to ban memes. This article states that “online content sharing service providers and right holders shall cooperate in good faith in order to ensure that unauthorised protected works or other subject matter are not available on their services.”

More or less, the article is saying that any websites that host large amounts of user-generated content (think YouTube, Twitter and Facebook) are responsible for taking down that content if it infringes on copyright. An earlier version of the Directive even referred to “proportionate content recognition technologies” which sounds a lot like asking platform owners to use automate filters to scan every piece of upload content and stop anything that might violate copyright from being uploaded.

Meme Filter

Since then, there is a horde of creating a meme filter that can classify most of the memes over the internet and at the same time create a meme filter exploiter that can create memes which the filters are unable to classify. We mainly focused on creating a meme filter in this project.

We create a meme filter by building an image classifier model. There are many approaches to classify an image like detecting a single picture of a cat or dog (eg doge meme or lolcats meme) in a picture or by detecting multiple image boundaries inside a single image (eg kids adults legends meme) or detect a single child's picture (eg gavin meme). Another approach is to detect image macros. We will discuss this topic in detail in the following sections. Later we will create such image classifier for meme classification.

How Image Classification Works - An Overview

Data-driven approach

How might we go about writing an algorithm that can classify images into distinct categories? Unlike writing an algorithm for, for example, sorting a list of numbers, it is not obvious how one might write an algorithm for identifying cats in images. Therefore, instead of trying to specify what every one of the categories of interest look like directly in code, the approach that we take is not unlike one you would take with a child: we provide the computer with many examples of each class and then develop learning algorithms that look at these examples and learn about the visual appearance of each class. This approach is referred to as a data-driven approach, since it relies on first accumulating a training dataset of labeled images. Figure 1 shows an example of what such a dataset might look like.

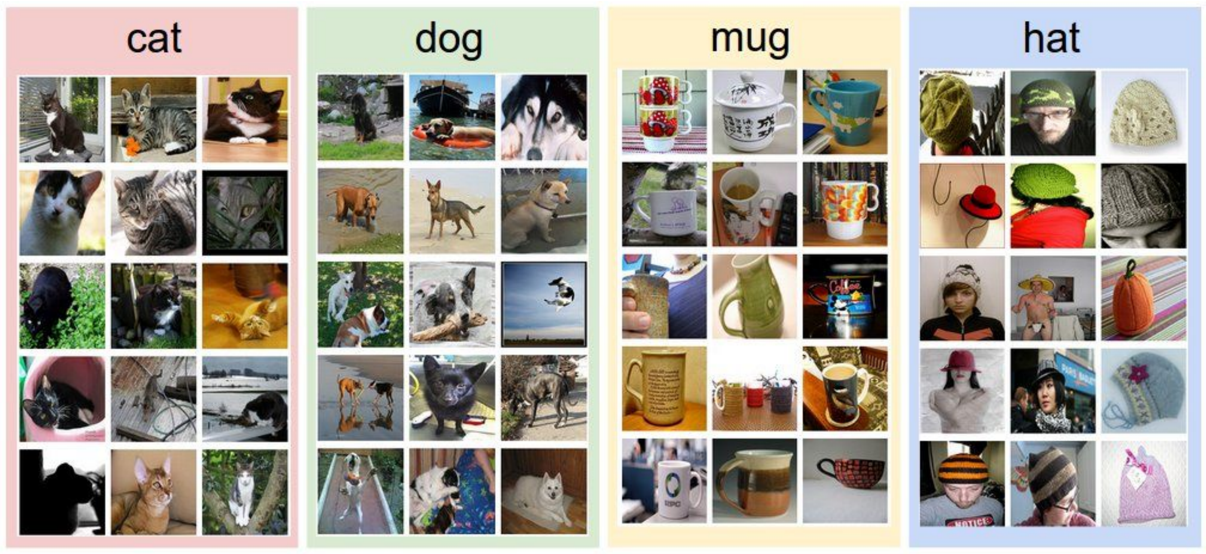


Figure 1: An example training set for four visual categories.

Figure only shows a couple of images for each category. In practice we may have thousands of categories and hundreds of thousands of images for each category.

The image classification pipeline

We've seen that the task in Image Classification is to take an array of pixels that represents a single image and assign a label to it. Our complete pipeline can be formalized as follows:

- **Input:** Our input consists of a set of N images, each labeled with one of K different classes. We refer to this data as the training set.
- **Learning:** Our task is to use the training set to learn what every one of the classes looks like. We refer to this step as training a classifier, or learning a model.
- **Evaluation:** In the end, we evaluate the quality of the classifier by asking it to predict labels for a new set of images that it has never seen before. We will then compare

the true labels of these images to the ones predicted by the classifier. Intuitively, we're hoping that a lot of the predictions match up with the true answers (which we call the ground truth).

Challenges in Image Classification

The task of recognizing a visual concept (e.g. cat) is relatively trivial for a human to perform, but it is worth considering the challenges involved from the perspective of a Computer Vision algorithm. As we present (an inexhaustive) list of challenges below, keep in mind the raw representation of images as a 3-D array of brightness values:

- **Viewpoint variation:** A single instance of an object can be oriented in many ways with respect to the camera.
- **Scale variation:** Visual classes often exhibit variation in their size (size in the real world, not only in terms of their extent in the image).
- **Deformation:** Many objects of interest are not rigid bodies and can be deformed in extreme ways.
- **Occlusion:** The objects of interest can be occluded. Sometimes only a small portion of an object (as little as few pixels) could be visible.
- **Illumination conditions:** The effects of illumination are drastic on the pixel level.
- **Background clutter:** The objects of interest may blend into their environment, making them hard to identify.
- **Intra-class variation:** The classes of interest can often be relatively broad, such as chair. There are many different types of these objects, each with their own appearance.

A good image classification model must be invariant to the cross product of all these variations, while simultaneously retaining sensitivity to the inter-class variations.

Image Classification using Neural Networks

With the recent advancements in the field of Neural Networks and Deep Learning, the task of image classification is solved mainly through these methods. We will first begin with discussing some concepts of neural networks and build upon these concepts to work towards our classifier.

Neural network

Neural Network is a machine learning algorithm, which is built on the principle of the organization and functioning of biological neural networks. This concept arose in an attempt to simulate the processes occurring in the brain by Warren McCulloch and Walter Pitts in 1943.

Neural networks consist of individual units called neurons. Neurons are located in a series of groups — layers (see figure allow). Neurons in each layer are connected to neurons of the next layer. Data comes from the input layer to the output layer along these compounds. Each individual node performs a simple mathematical calculation. Then it transmits its data to all the nodes it is connected to.

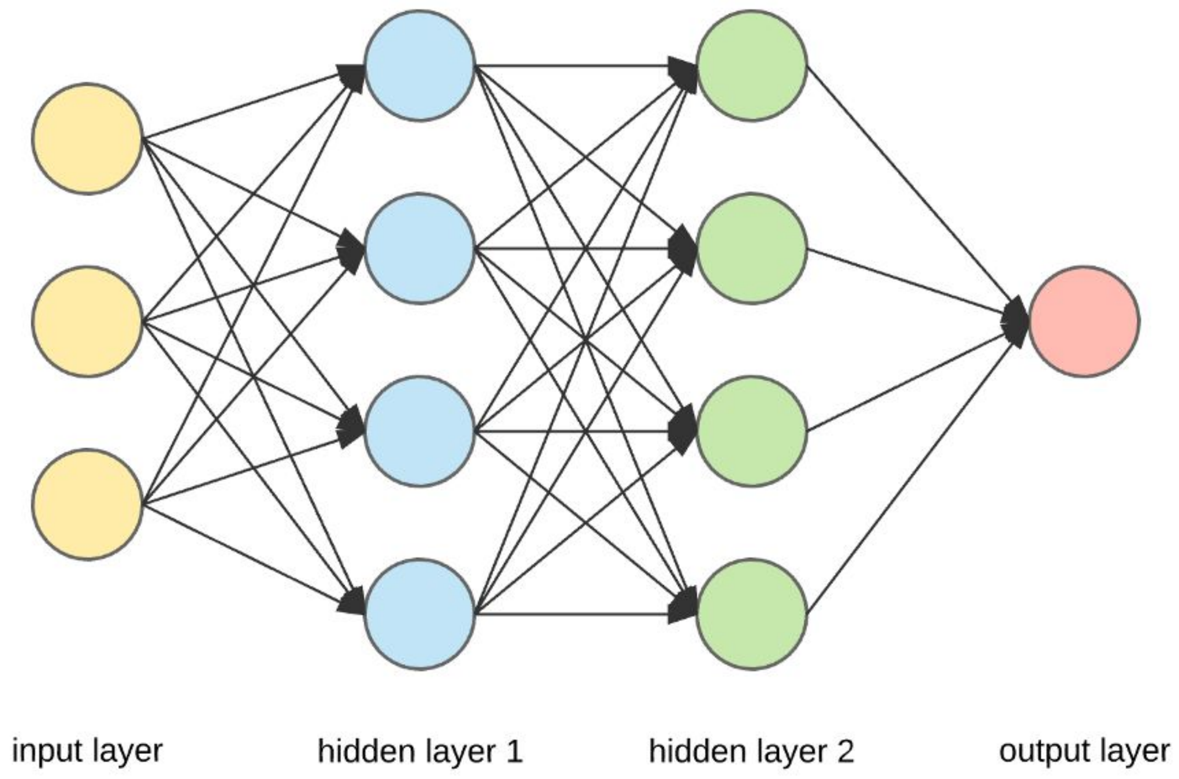


Figure 2: Illustration of Neural Network

The last wave of neural networks came in connection with the increase in computing power and the accumulation of experience. That brought Deep learning, where technological structures of neural networks have become more complex and able to solve a wide range of tasks that could not be effectively solved before. Image classification is a prominent example.

Convolutional neural networks and image classification

Convolutional neural networks (CNN) is a special architecture of artificial neural networks, proposed by Yann LeCun in 1988. CNN uses some features of the visual cortex. One of the most popular uses of this architecture is image classification. For example Facebook uses CNN for automatic tagging algorithms, Amazon — for generating product recommendations and Google — for search through among users' photos.

Let us consider the use of CNN for image classification in more detail. The main task of image classification is acceptance of the input image and the following definition of its class. This is a skill that people learn from their birth and are able to easily determine that the image in the Figure 3 (a) is an elephant. But the computer sees the pictures quite differently as in Figure 3 (b).

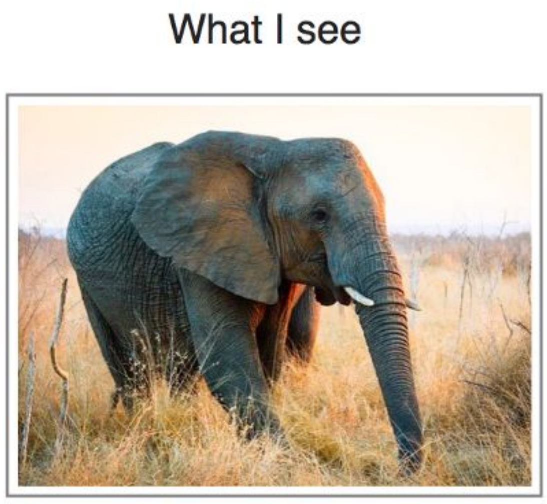
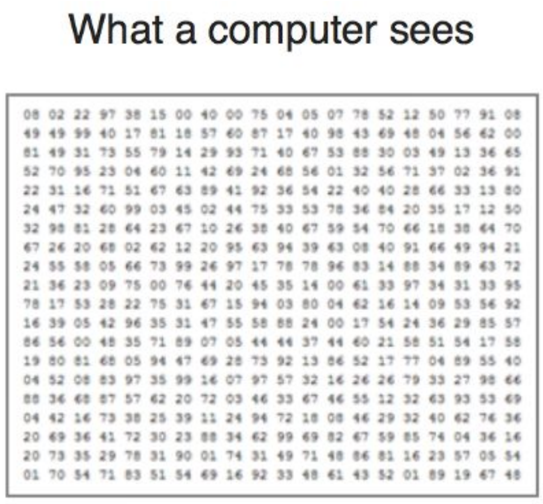


Figure 3a: An image of elephant (human perspective)


What a computer sees

08	02	22	97	38	15	00	40	00	75	04	05	07	78	52	12	50	77	91	08
49	49	99	40	17	81	18	57	60	87	17	40	98	43	69	48	04	56	62	00
81	49	31	73	55	79	14	29	93	71	40	67	53	88	30	03	49	13	36	65
52	70	95	23	04	60	11	42	69	24	68	56	01	32	56	71	37	02	36	91
22	31	16	71	51	67	63	89	41	92	36	54	22	40	40	28	66	33	13	80
24	47	32	60	99	03	45	02	44	75	33	53	78	36	84	20	35	17	12	50
32	98	81	28	64	23	67	10	26	38	40	67	59	54	70	66	18	38	64	70
67	26	20	68	02	62	12	20	95	63	94	39	63	08	40	91	66	49	94	21
24	55	58	05	66	73	99	26	97	17	78	78	96	83	14	88	34	89	63	72
21	36	23	09	75	00	76	44	20	45	35	14	00	61	33	97	34	31	33	95
78	17	53	28	22	75	31	67	15	94	03	80	04	62	16	14	09	53	56	92
16	39	05	42	96	35	31	47	55	58	88	24	00	17	54	24	36	29	85	57
86	56	00	48	35	71	89	07	05	44	44	37	44	60	21	58	51	54	17	58
19	80	81	68	05	94	47	69	28	73	92	13	86	52	17	77	04	89	55	40
04	52	08	83	97	35	99	16	07	97	57	32	16	26	26	79	33	27	98	66
88	36	68	87	57	62	20	72	03	46	33	67	46	55	12	32	63	93	53	69
04	42	16	73	38	25	39	11	24	94	72	18	08	46	29	32	40	62	76	36
20	69	36	41	72	30	23	88	34	62	99	69	82	67	59	85	74	04	36	16
20	73	35	29	78	31	90	01	74	31	49	71	48	86	81	16	23	57	05	54
01	70	54	71	83	51	54	69	16	92	33	48	61	43	52	01	89	19	67	48

Figure 3b: An image of elephant (computer perspective)

Instead of the image, the computer sees an array of pixels. For example, if image size is 300 x 300. In this case, the size of the array will be 300x300x3. Where 300 is width, next 300 is height and 3 is RGB channel values. Then the computer assign a value from 0 to 255 to each of these pixels. This value describes the intensity of the pixel at each point.

To solve this problem the computer looks for the characteristics of the base level. In human understanding such characteristics are for example the trunk or large ears. For the computer, these characteristics are boundaries or curvatures. And then through the groups of convolutional layers the computer constructs more abstract concepts.

To grab these abstract concepts of image by the computer, the image is passed through a series of convolutional, nonlinear, pooling layers and fully connected layers, and then generates the output. The Convolution layer is always the first. The image (matrix with pixel values) is entered into it. Imagine that the reading of the input matrix begins at the top left of image. Next the software selects a smaller matrix there, which is called a filter (or neuron, or core). Then the filter produces convolution, i.e. moves along the input image. The filter's task is to multiply its values by the original pixel values. All these multiplications are summed up. One number is obtained in the end. Since the filter has read the image only in the upper left corner, it moves further and further right by 1 unit performing a similar operation. After passing the filter across all positions, a matrix is obtained, but smaller than a input matrix. Figure 4 illustrate the above mentioned algorithm.

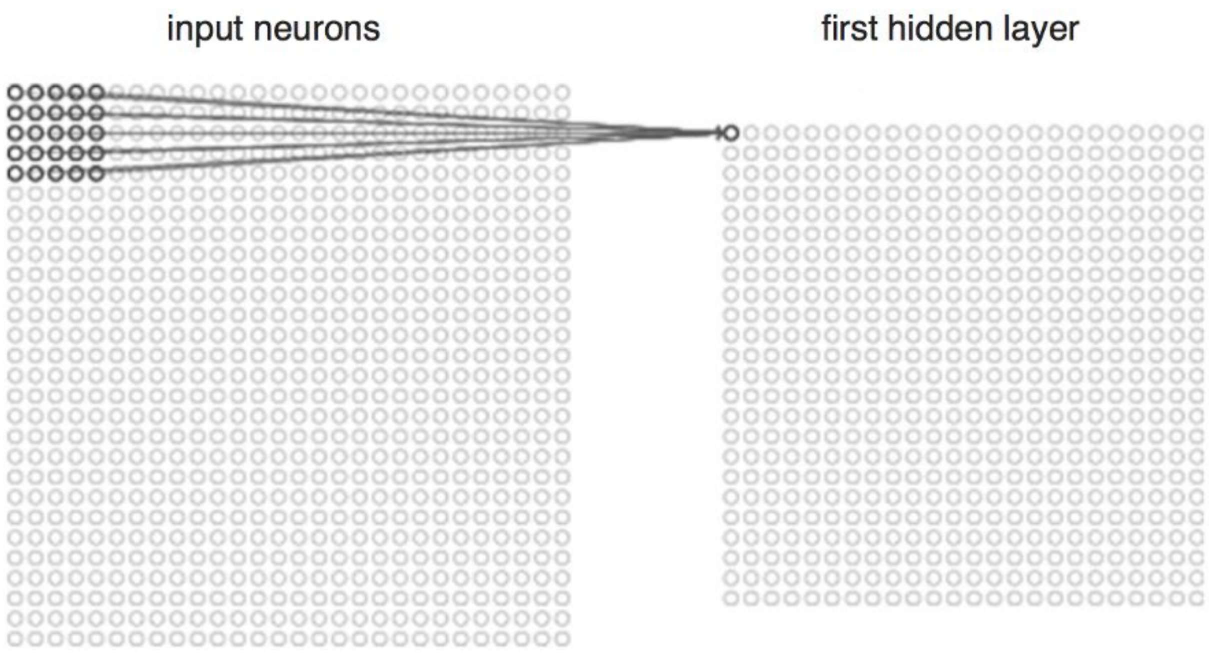


Figure 4: Illustration of CNN filter

This operation, from a human perspective, is analogous to identifying boundaries and simple colours on the image. But in order to recognize the properties of a higher level such as the trunk or large ears the whole network is needed.

The network will consist of several convolutional networks mixed with nonlinear and pooling layers. When the image passes through one convolution layer, the output of the first layer becomes the input for the second layer. And this happens with every further convolutional layer.

The nonlinear layer is added after each convolution operation. It has an activation function, which brings nonlinear property. Without this property a network would not be sufficiently intense and will not be able to model the response variable (as a class label).

The pooling layer follows the nonlinear layer. It works with width and height of the image and performs a downsampling operation on them. As a result the image volume is reduced. This means that if some features (as for example boundaries) have already been identified in the previous convolution operation, than a detailed image is no longer needed for further processing, and it is compressed to less detailed pictures.

After completion of series of convolutional, nonlinear and pooling layers, it is necessary to attach a fully connected layer. This layer takes the output information from convolutional networks. Attaching a fully connected layer to the end of the network results in an N dimensional vector, where N is the amount of classes from which the model selects the desired class

How Convolutional NN Works

Instead of feeding entire images into our neural network as one grid of numbers, we're going to do something a lot smarter that takes advantage of the idea that an object is the same no matter where it appears in a picture.

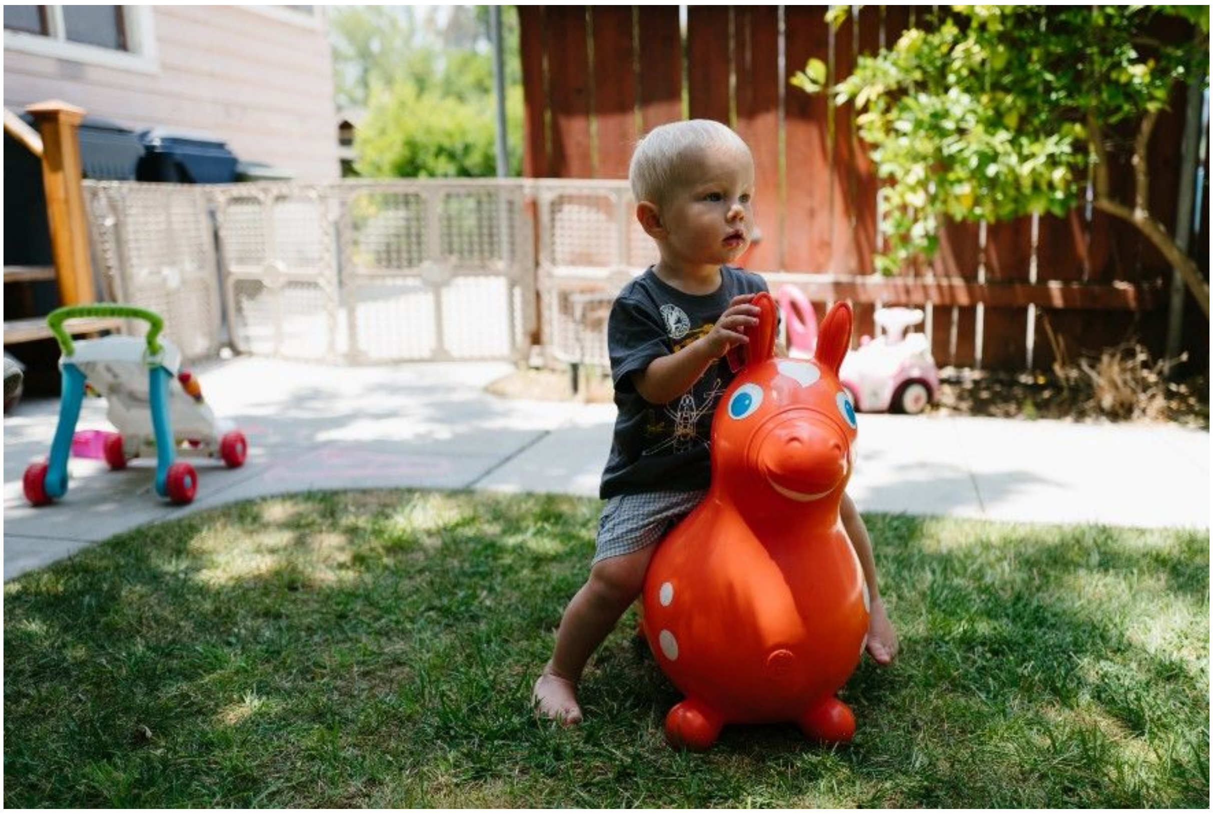


Figure 5: Example Image

Here's how it's going to work. Let's take a sample image, Figure 5, and apply CNN to it step by step —

Step 1: Break the image into overlapping image tiles

Similar to our sliding filter approach above, let's pass a sliding window over the entire original image and save each result as a separate, tiny picture tile as shown in Figure 6.

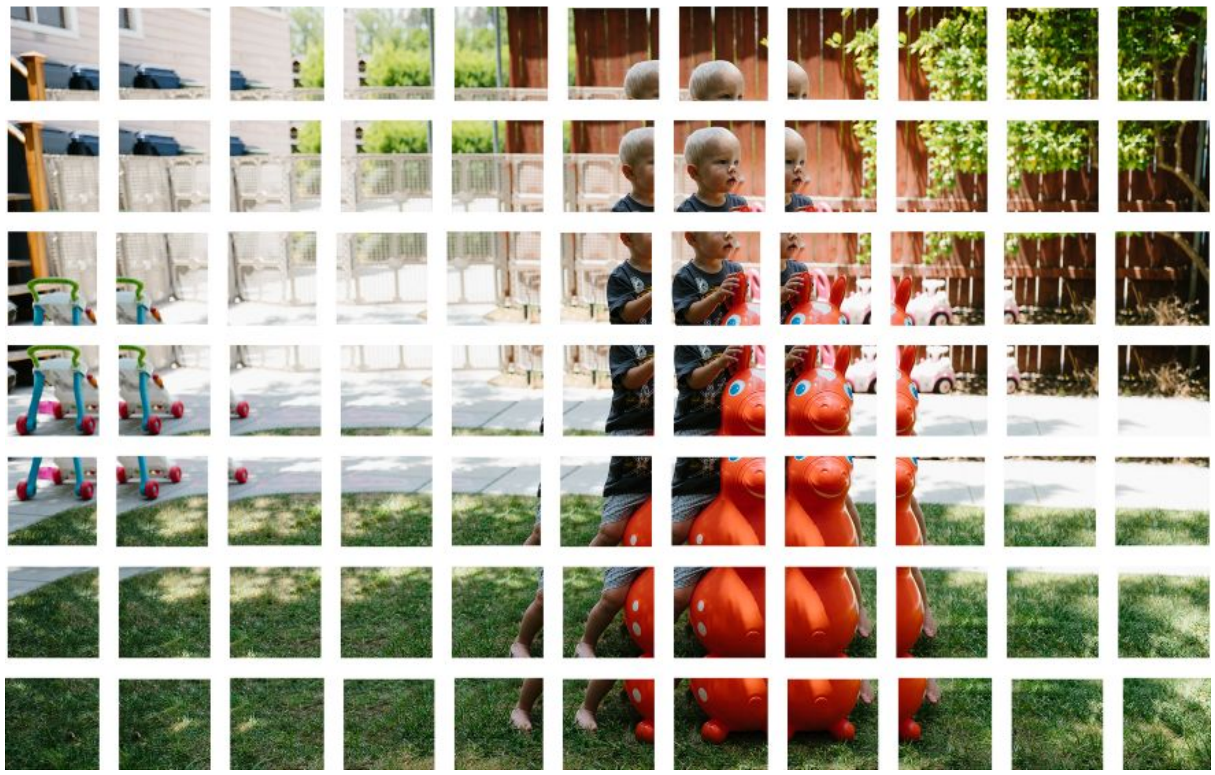


Figure 6: Framed Image

By doing this, we turned our original image into 77 equally-sized tiny image tiles.

Step 2: Feed each image tile into a small neural network

Earlier, we fed a single image into a neural network to classify images. We'll do the exact same thing here, but we'll do it for each individual image tile. Illustration is given in Figure 7.

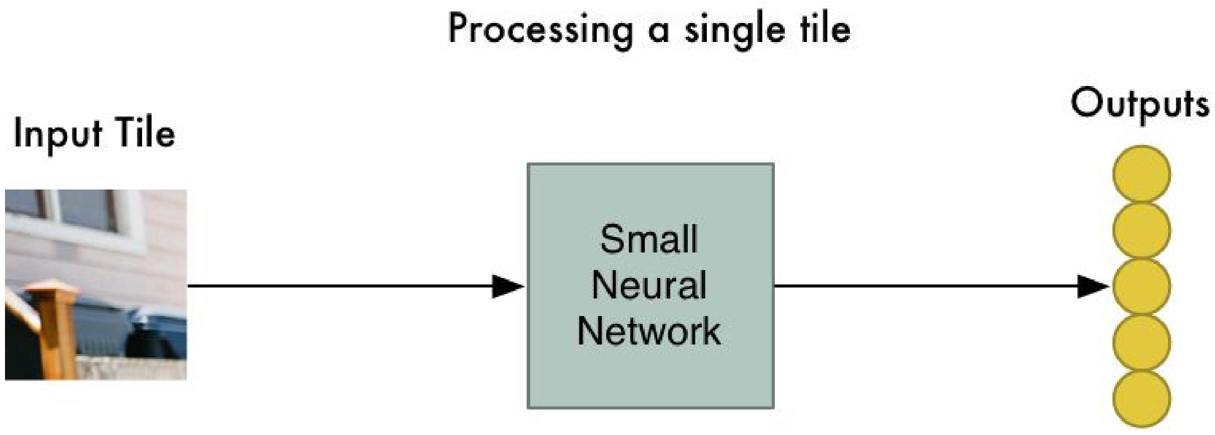


Figure 7: Illustration of NN on image tile

However, there's one big twist: We'll keep the same neural network weights for every single tile in the same original image. In other words, we are treating every image tile equally. If something interesting appears in any given tile, we'll mark that tile as interesting.

Step 3: Save the results from each tile into a new array

We don’t want to lose track of the arrangement of the original tiles. So we save the result from processing each tile into a grid in the same arrangement as the original image. It looks like Figure 8.

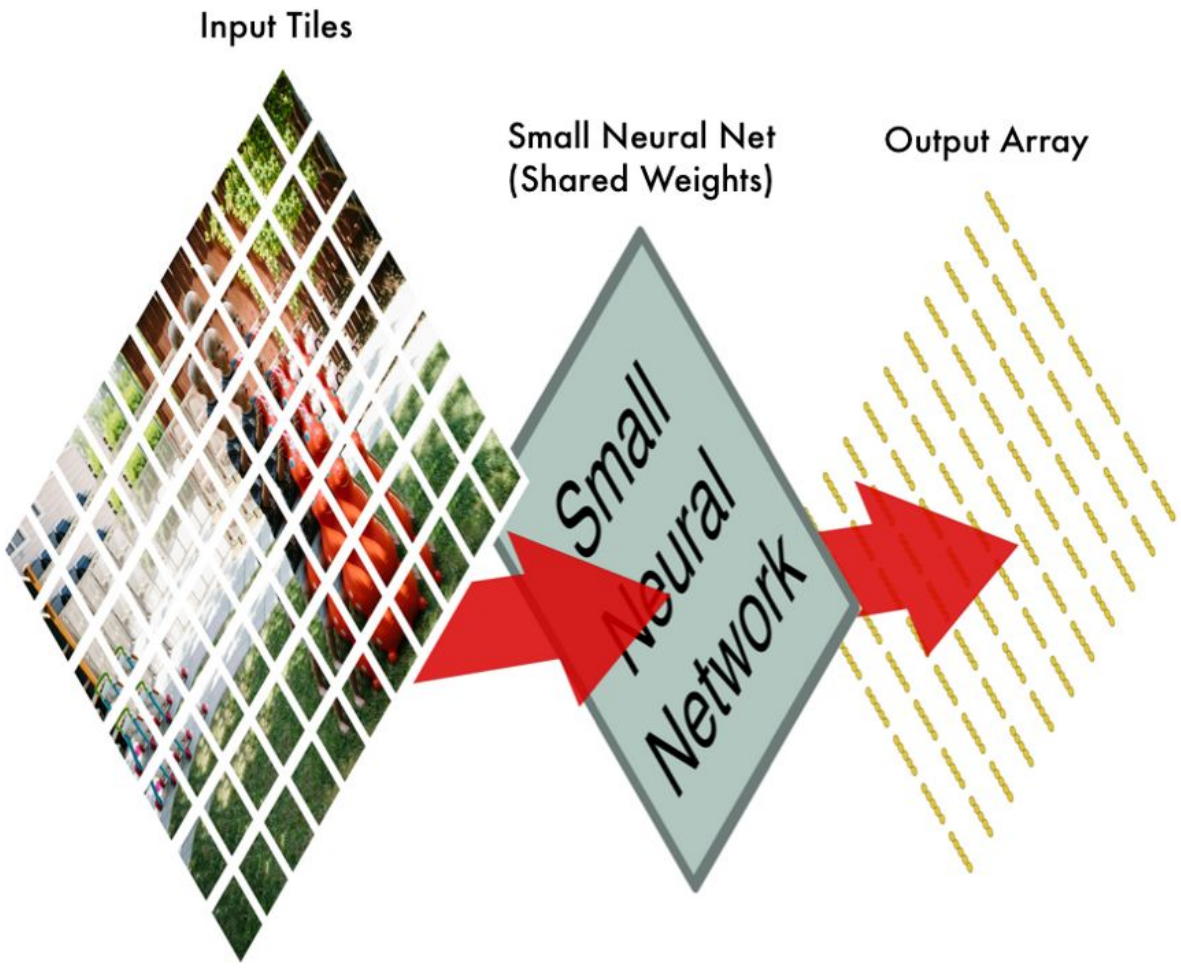


Figure 8: Saving the results in an array

In other words, we’ve started with a large image and we ended with a slightly smaller array that records which sections of our original image were the most interesting.

Step 4: Downsampling

The result of Step 3 was an array that maps out which parts of the original image are the most interesting. But that array is still pretty big as shown in Figure 9.

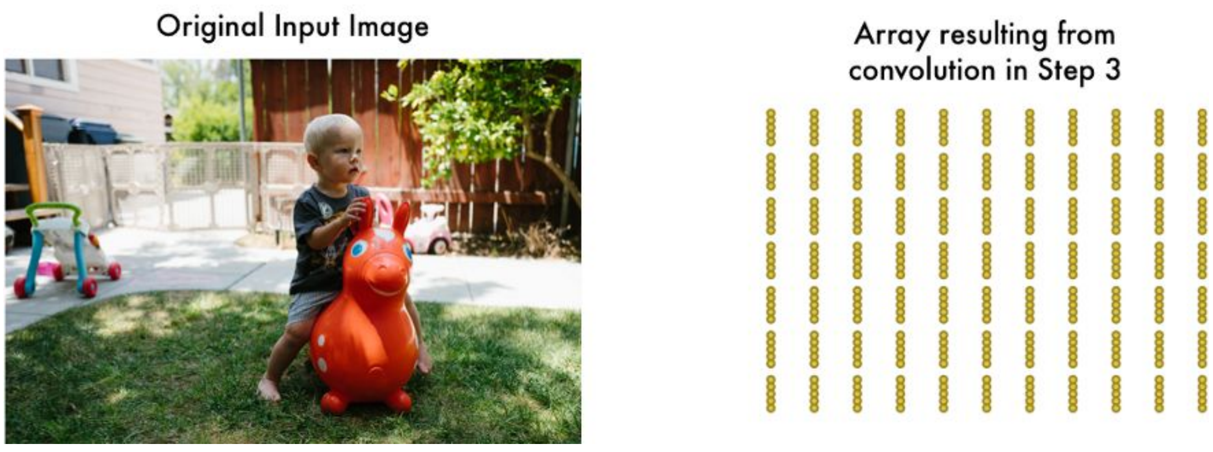


Figure 9: Illustration of saved array

To reduce the size of the array, we downsample it using an algorithm called max pooling. In max pooling, we'll just look at each 2x2 square of the array and keep the biggest number. Illustration is given in Figure 10.

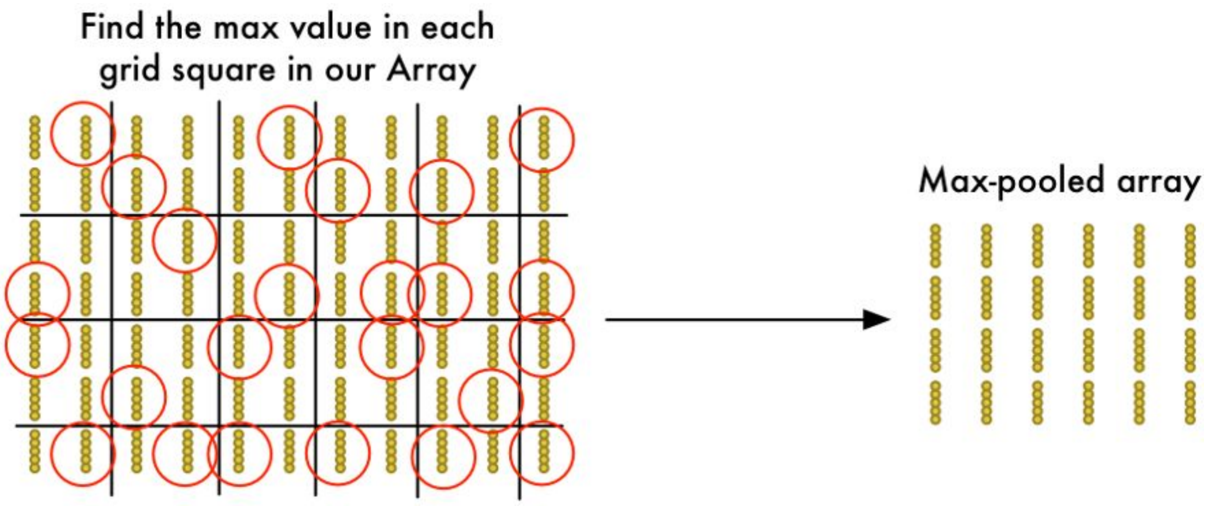


Figure 10: Creating max-pooled array

The idea here is that if we found something interesting in any of the four input tiles that makes up each 2x2 grid square, we'll just keep the most interesting bit. This reduces the size of our array while keeping the most important bits.

Final step: Make a prediction

So far, we've reduced a giant image down into a fairly small array. Guess what? That array is just a bunch of numbers, so we can use that small array as input into another neural network. This final neural network will decide if the image is or isn't a match. To differentiate it from the convolution step, we call it a "fully connected" network.

So from start to finish, our whole five-step pipeline looks like Figure 11.

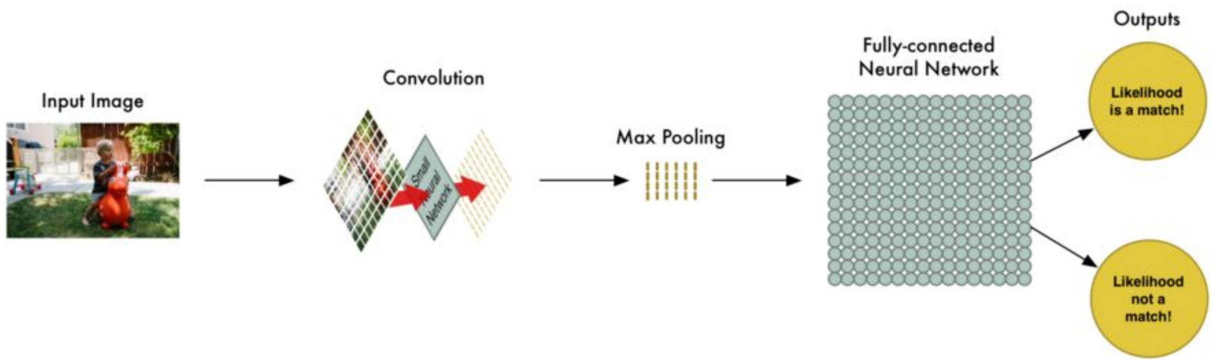


Figure 11: All steps of CNN

Adding Even More Steps

Our image processing pipeline is a series of steps: convolution, max-pooling, and finally a fully-connected network.

When solving problems in the real world, these steps can be combined and stacked as many times as you want! You can have two, three or even ten convolution layers. You can throw in max pooling wherever you want to reduce the size of your data.

The basic idea is to start with a large image and continually boil it down, step-by-step, until you finally have a single result. The more convolution steps you have, the more complicated features your network will be able to learn to recognize.

For example, the first convolution step might learn to recognize sharp edges, the second convolution step might recognize beaks using it’s knowledge of sharp edges, the third step might recognize entire birds using it’s knowledge of beaks, etc.

Figure 12 shows a more realistic deep convolutional network (like you would find in a research paper).

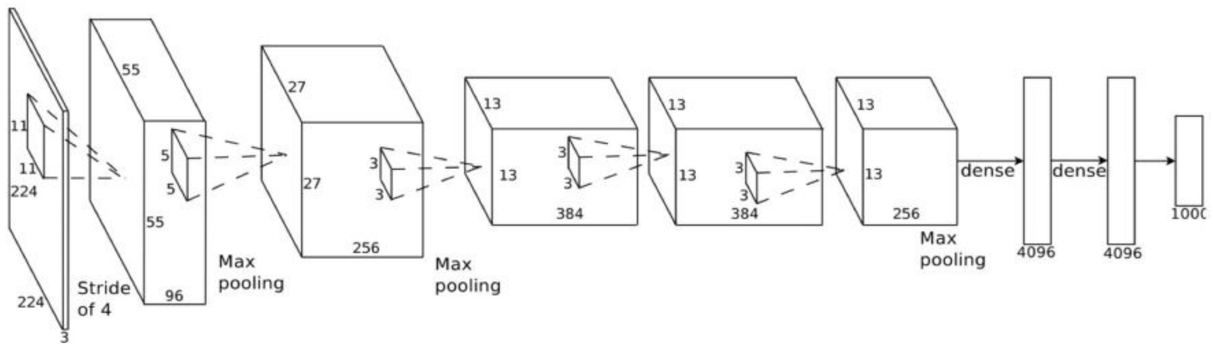


Figure 12: Realistic Illustration of CNN

In this case, they start a 224 x 224 pixel image, apply convolution and max pooling twice, apply convolution 3 more times, apply max pooling and then have two fully-connected layers. The end result is that the image is classified into one of 1000 categories!

MEME Classifier

Now that we have some working knowledge of image classification, we now proceed with creating our own MEME classifier.

First step is to add all the required libraries.

```
#set the current working directory according to requirements
import os
os.chdir('.../your_working_directory_path/')
import cv2
import tqdm
from random import shuffle
from keras.models import Sequential,model_from_json
from keras.layers import *
from keras.optimizers import *
import numpy as np
import matplotlib.pyplot as plt
# from tensorflow.python.keras.applications.resnet50 import
preprocess_input
# from tensorflow.python.keras.preprocessing.image import load_img,
img_to_array
```

Next step is just an optional step to make our life easier. Here we just create a path variable so that we can just use the variable instead of writing the whole path.

```
#loading images/preprocessing them
path1='memes/'
path2='memesnt/'
path3='test/'
# imgs=os.listdir(path1)
```

Next step is a very important step. Here we introduce a new term called ‘One-Hot Encoding’. One-hot encoding is used on categorical data to prepare the data before fitting it to a model. Categorical data are variables that contain label values rather than numeric values. The number of possible values is often limited to a fixed set. Some examples include:

- A “pet” variable with the values: “dog” and “cat“.
- A “color” variable with the values: “red“, “green” and “blue“.
- A “place” variable with the values: “first”, “second” and “third“.

Each value represents a different category. Some categories may have a natural relationship to each other, such as a natural ordering. The “place” variable above does have a natural ordering of values. This type of categorical variable is called an ordinal variable.

Some algorithms can work with categorical data directly. For example, a decision tree can be learned directly from categorical data with no data transform required (this depends on the specific implementation). But many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric.

In general, this is mostly a constraint of the efficient implementation of machine learning algorithms rather than hard limitations on the algorithms themselves. This means that categorical data must be converted to a numerical form. If the categorical variable is an output variable, you may also want to convert predictions by the model back into a categorical form in order to present them or use them in some application.

One such way is One-Hot Encoding.

One-Hot Encoding

For categorical variables where no such ordinal relationship exists, the integer encoding is not enough. In fact, using this encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories).

In this case, a one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value.

In the “color” variable example, there are 3 categories and therefore 3 binary variables are needed. A “1” value is placed in the binary variable for the color and “0” values for the other colors.

For example:

```
red         green       blue
1           0           0
0           1           0
0           0           1
```

We do a similar thing with our image dataset. We create a function to label all the images which are MEMEs with [1,0] and all other image with [0,1].

```
#A one hot encoding function
def one_hot_encoder(img):
    label=img.split('.')[0]
#     path=os.path(img).split('/')[-1]
    if label.isalnum():
        ohl=np.array([1,0])
    else:
        ohl=np.array([0,1])
    return ohl
```

Next we define a feature extractor function. This function first takes all the images in our training dataset and resize them into a standard format of 64 x 64 pixel. Then it appends the images in our `train_images` variable along with the label. In the end, it shuffles all the images to randomise our dataset.

Here, we would like to make a special note about the **importance of shuffling** the dataset we introduce to our model. The order that the observations are exposed to the model affects internal decisions. Some algorithms are especially susceptible to this, like neural networks. It is good practice to randomly shuffle the training data before each training iteration. Even if your algorithm is not susceptible. It's a best practice.

```
def feature_extractor():
    train_images=[]
    for i in tqdm.tqdm(os.listdir(path1)):
        path=os.path.join(path1,i)
        img=cv2.imread(path,cv2.IMREAD_GRAYSCALE)
        img=cv2.resize(img,(64,64))
#         print(img)
#         train_images.append([np.array(img),one_hot_encoder(i)])
        train_images.append([np.array(img),np.array([1,0])])
    for i in tqdm.tqdm(os.listdir(path2)):
        path=os.path.join(path2,i)
        img=cv2.imread(path,cv2.IMREAD_GRAYSCALE)
        img=cv2.resize(img,(64,64))
        train_images.append([np.array(img),np.array([0,1])])
    shuffle(train_images)
    return train_images
```

In the next step, we create another function to read the training data from the dataset and apply our One-Hot Encoding function to it.

```
def test_data_reader():
    test_images=[]
    for i in tqdm.tqdm(os.listdir(path3)):
        path=os.path.join(path3,i)
        img=cv2.imread(path,cv2.IMREAD_GRAYSCALE)
        img=cv2.resize(img,(64,64))
        test_images.append([np.array(img),one_hot_encoder(i)])
    return test_images
```

Next, we make use of the 2 functions we created before.

```
training_images = feature_extractor()

tr_img_data = np.array([i[0] for i in
training_images]).reshape(-1,64,64,1)
tr_lbl_data = np.array([i[1] for i in training_images])
```

```
100%|██████████████████████████████████████████████████████████████████
████| 3120/3120 [01:43<00:00, 30.10it/s]
100%|██████████████████████████████████████████████████████████████████
███| 2340/2340 [00:22<00:00, 102.85it/s]
```

```
testing_images = test_data_reader()
tst_img_data = np.array([i[0] for i in
testing_images]).reshape(-1,64,64,1)
tst_lbl_data = np.array([i[1] for i in testing_images])
```

```
100%|██████████████████████████████████████████████████████████████████
██████| 374/374 [00:07<00:00, 47.82it/s]
```

Now comes the time when we design our actual model. As we explained earlier, our image processing pipeline is a series of steps: convolution, max-pooling, and finally a fully-connected network. Multiple layers of this pipeline are passed upon the images in the dataset. So how do you know how many steps you need to combine to make your image classifier work?
Honestly, you have to answer this by doing a lot of experimentation and testing. You might have to train 100 networks before you find the optimal structure and parameters for the problem you are solving. Machine learning involves a lot of trial and error!

```
model=Sequential()

model.add(InputLayer(input_shape=[64,64,1]))
model.add(Conv2D(filters=32,kernel_size=5,strides=1,padding='same',activ
ation='relu'))
model.add(MaxPool2D(pool_size=5,padding='same'))

model.add(Conv2D(filters=50,kernel_size=5,strides=1,padding='same',activ
ation='relu'))
model.add(MaxPool2D(pool_size=5,padding='same'))
model.add(Conv2D(filters=80,kernel_size=5,strides=1,padding='same',activ
ation='relu'))
model.add(MaxPool2D(pool_size=5,padding='same'))

model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512,activation='relu'))
model.add(Dropout(0.5))


model.add(Dense(2,activation='softmax'))
optimizer=Adam(lr=1e-3)

model.compile(optimizer=optimizer,loss='categorical_crossentropy',metric
s=['accuracy'])
model.fit(x=tr_img_data,y=tr_lbl_data,epochs=50,batch_size=100)
model.summary()
```

Now, the last step is to validate our model and visualize our result. We run our model on random images in the testing dataset and check their result.

```
fig=plt.figure(figsize=(14,14))
for cnt,data in enumerate(testing_images[190:220]):
    y=fig.add_subplot(6,5,cnt+1)
    img=data[0]
    data=img.reshape(1,64,64,1)
    model_out=model.predict([data])

    if np.argmax(model_out)==1:
        str_label='Not a Meme'
    else:
        str_label='Meme'

    y=plt.imshow(img,cmap='gray')
    plt.title(str_label)
    y.axes.get_xaxis().set_visible(False)
    y.axes.get_yaxis().set_visible(False)
```

Results

We were successfully able to build a model that is able to distinguish macro memes made by people from non meme images and even meme templates. The Figure 13 shows the results of our classifier.

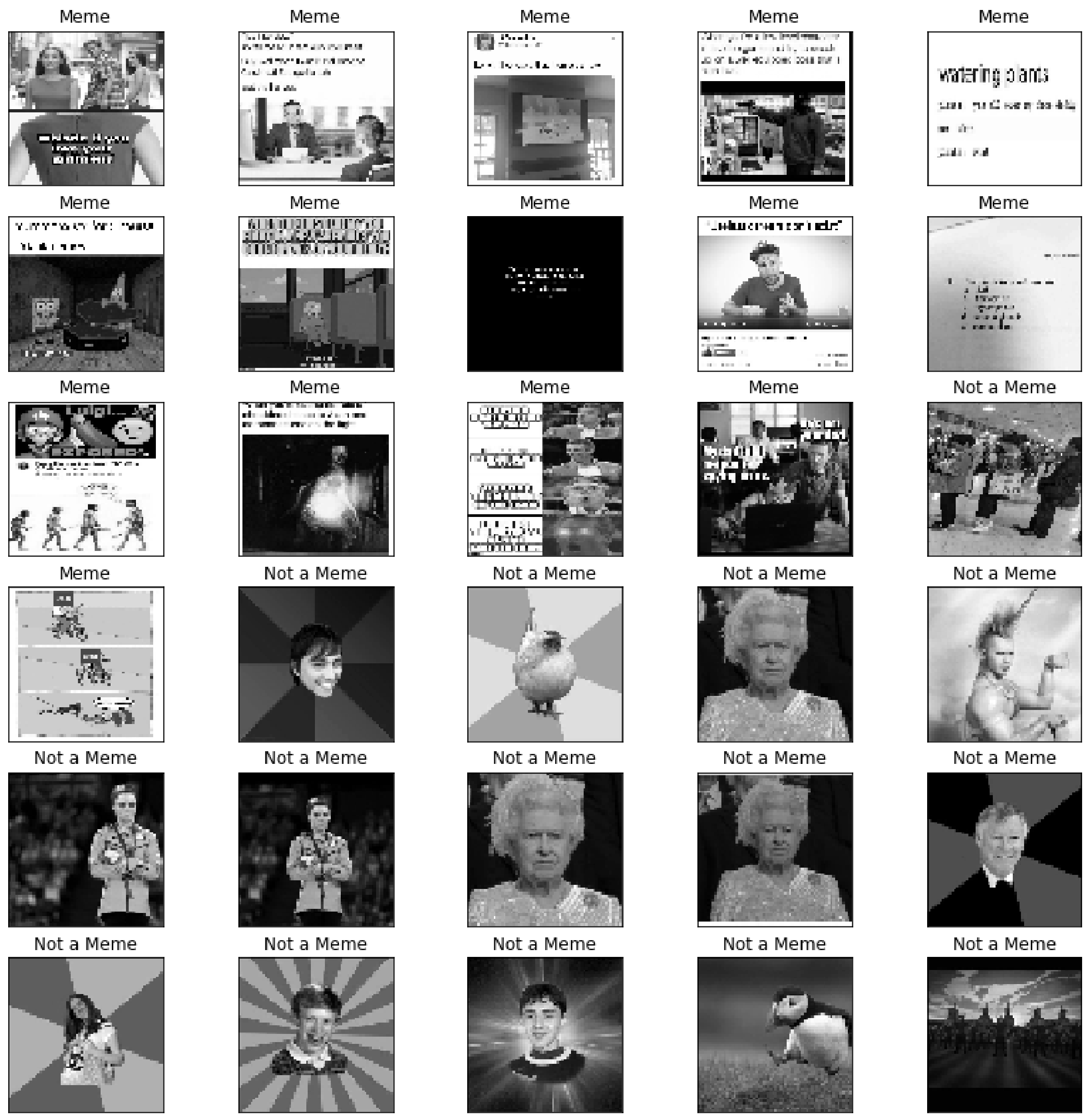


Figure 13: Results of our Meme Classifier

Conclusion

The upload filter developed by us is a basic classifier and is still in development phase. An actual deployable filter would include a lot more functionalities. The real task is to make a filter that while ensuring enforcement of copyright laws doesn't cease the creativity of people. The task thus includes identifying all forms of content and classifying them correctly. To make such a state of the art filter a lot of time and money would be spent by software companies. If apt measures are not taken it could be the end of meme making as we know it.

References

1. Dawkins, Richard (1989), The Selfish Gene (2 ed.), Oxford University Press, p. 192, ISBN 978-0-19-286092-7, We need a name for the new replicator, a noun that conveys the idea of a unit of cultural transmission, or a unit of imitation. 'Mimeme' comes from a suitable Greek root, but I want a monosyllable that sounds a bit like 'gene'. I hope my classicist friends will forgive me if I abbreviate mimeme to meme. If it is any consolation, it could alternatively be thought of as being related to 'memory', or to the French word même. It should be pronounced to rhyme with 'cream'.
2. Millikan 2004, p. 16; Varieties of meaning. "Richard Dawkins invented the term 'memes' to stand for items that are reproduced by imitation rather than reproduced genetically."
3. Peirson V, Tolunay(2017), Dank Learning: Generating memes using deep neural networks, We introduce a novel meme generation system, which given any image can produce a relevant and humorous caption.
4. https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html
5. https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8