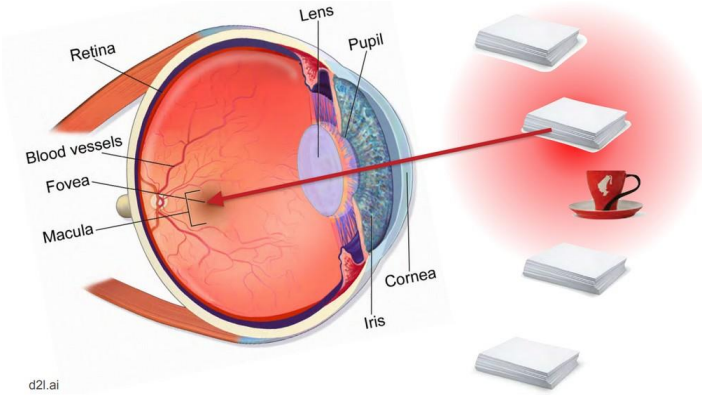
Computer Vision with Attention Model

Table of Content

Introduction

Modern day techniques employed in Neural Networks in the domain of Computer Vision include "Attention Mechanisms". Computer Vision has aimed to emulate human visual perception in terms of code-based algorithms. Before diving further into the Attention Mechanisms used in Computer Vision, let's take a brief look at how attention mechanisms are embedded in and inspired by human visual capabilities.

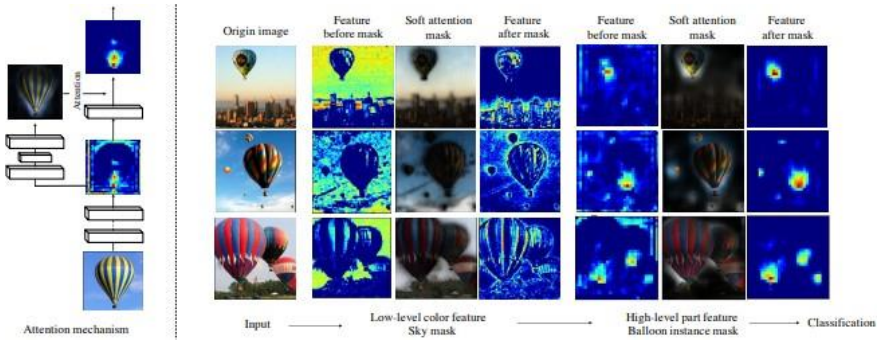


Light traverses from the object of interest (which in this case is a leaf) to the Macula, which is the primary functional region of the Retina inside the eye. However, what happens when we have multiple objects in the field of view? When we must focus on a single object when there is an array of diverse objects in our field of view, the attention mechanism within our visual perception system uses a sophisticated group of filters to create a blurring effect (similar to that of "Bokeh" in digital photography) so the object of interest is in focus, while the surrounding is faded or blurred.

Now, back to Attention Mechanisms in Deep Learning.

The attention mechanism has become a very popular technique in natural language processing, image processing, and computer vision. The attention mechanism can generate attention-aware features and features that can be extracted based on spatial, context, or channel aware-features.

Using visual attention in an image classification task helps determine important image regions and their correlations. The presence or absence of image regions is critical to classification.

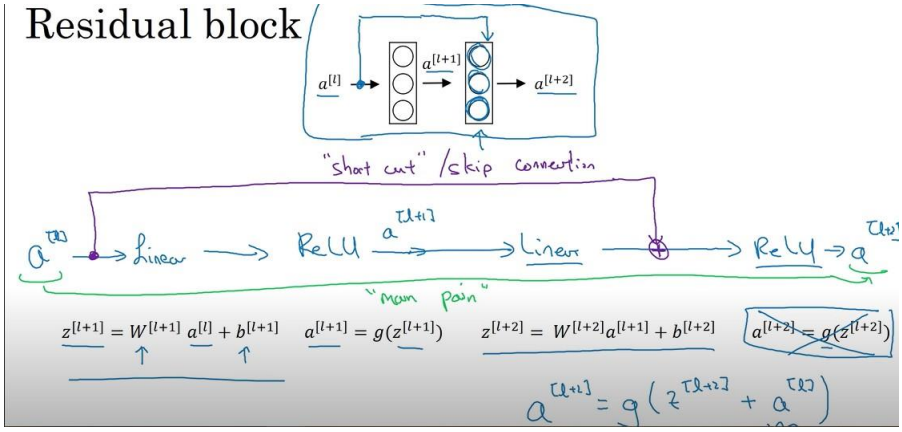


As we can see here in this image how the attention module focuses on a particular feature from layer to layer during the training of the model.

Residual Block

In order to learn about the model architecture, we need to first know about residual blocks and how do theywork?

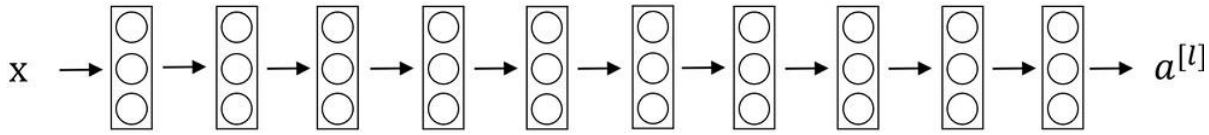
When we work with deeper neural networks it is hard to train the model because we should deal a lot with vanishing/exploding gradient descent and here I will try to explain how we can take activation functions from one layer and feed to a deeper layer in order to deal with deeper neural networks.



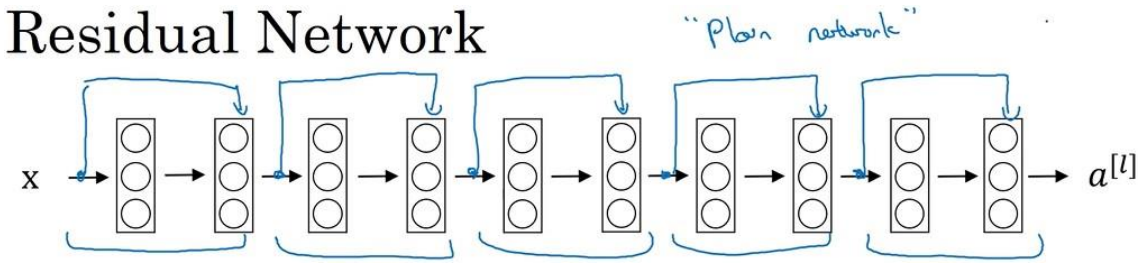
So, the main thing to understand here is that the information from one layer of an activation function can be passed deep into the neural network for training they are usually known as **shortcut/skip connection.**

Why do we do this and why is it required?

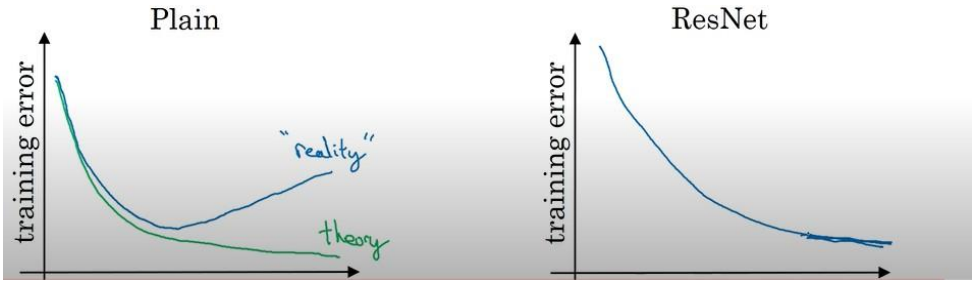
The inventors of the residual block architecture discovered that this technique is more useful in training deeper neural networks and a ResNet is formed by stacking many residual blacks together and forming a deeper network.



This is a plain neural network and to turn it into an ResNet we should add all those skip connections/shortcuts.



Now this network has 5 residual blocks stacked together and this is a residual network. While passing the activation function much deeper in the neural network helps with the vanishing/exploding gradient descent problem.



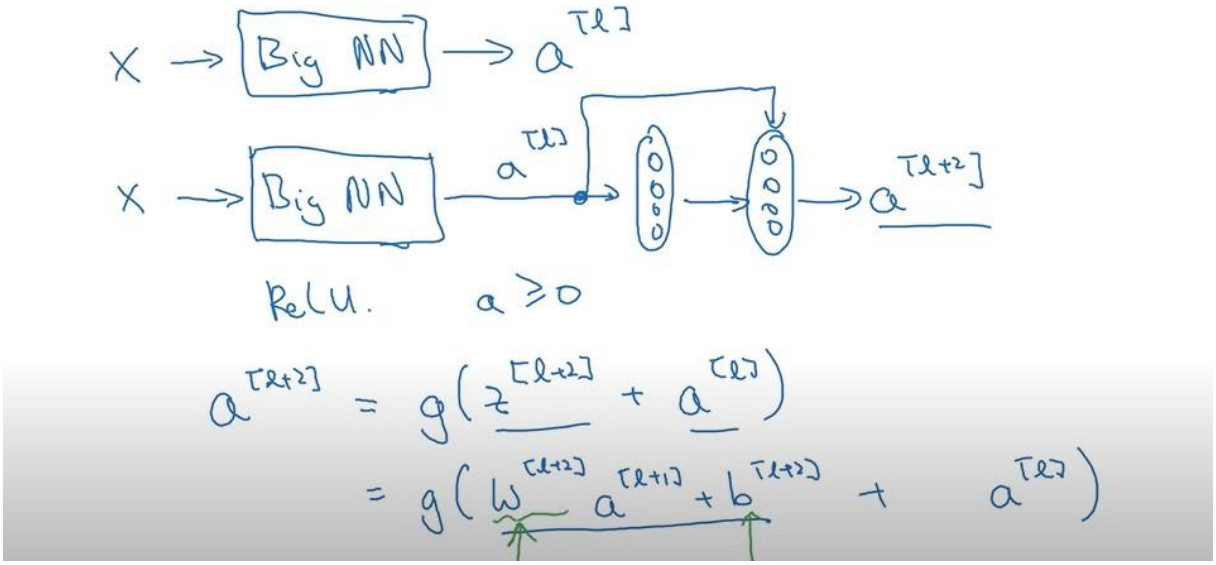
Plain Neural Network Vs Residual Neural Network

In the above image on the left side, we can see that using plain neural network the training error first decreases and then starts to increase, but in theory it says that as the network gets deeper and deeper the training error should decrease. In the right side we use ResNet and we can see that the training error continuously decreases and flattens after sometime.

Proof

Why do they work?

So, let's take an example of a plain neural network and make it deeper by adding two layers to it.



Explanation

- The activation function for the last layer would be ($a[l + 2] = g(z[l + 2] + a[l])$ for the last layer which includes the activation function of the first layer also(residual block). Now after expanding the term ($z[l + 2] = w[l + 2] * a[l + 1] + b[l + 2]$) ((weights*activation of previous layer) + bias).
- When we apply **L2 regularization** for weight decaying and sometimes for bias too now let's assume that after decaying the weights and bias have turned to 0 ($w[l + 2] = 0$, $b[l + 2] = 0$).
- Then the term g(w[l+2]*a[l+1]+b[l+2]) becomes 0 making the equation $g(a[l]) = 0 \Rightarrow a[l]$.

The main drawback of plain neural networks is that when they get deeper it gets hard of the neural network to learn new features and choose parameters from and this is why it makes our result worse than making it better in deeper neural networks.

The residual blocks make sure that the performance doesn't drop because we give it an activation function from above layers and when we apply gradient descent to it then the performance only gets better.

One thing to remember when we use residual block is that we need to make sure that the dimensions of the activation function from previous layer is same as that of the layer that we are apply it to. This is the reason we see a lot of **"same"** convolutions to make sure that the dimensions stay the same.

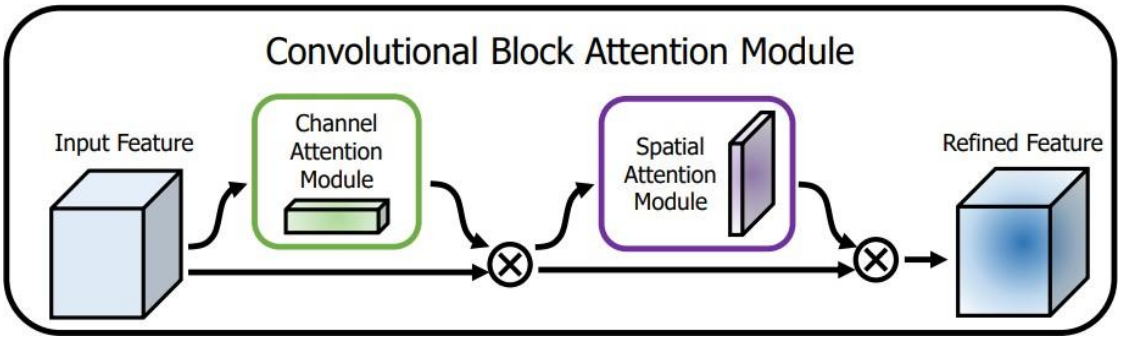
Code for Residual Block

```
def residual_block(input, input_channels=None, output_channels=None, kernel_size=(3, 3), stride=1):
    """
    full pre-activation residual block
    https://arxiv.org/pdf/1603.05027.pdf
    """
    if output_channels is None:
        output_channels = input.get_shape()[-1]
    if input_channels is None:
        input_channels = output_channels // 4

    strides = (stride, stride)

    x = BatchNormalization()(input)
    x = Activation('relu')(x)
    x = Conv2D(input_channels, (1, 1))(x)

    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(input_channels, kernel_size, padding='same', strides=stride)(x)

    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(output_channels, (1, 1), padding='same')(x)

    if input_channels != output_channels or stride != 1:
        input = Conv2D(output_channels, (1, 1), padding='same', strides=strides)(input)

    x = Add()([x, input])
    return x
```

What is an Attention Block?

Attention modules are used to make CNN learn and focus more on the important information, rather than learning non-useful background information. In the case of object detection, useful information is the objects or target class crop that we want to classify and localize in an image.

So, basically, we have 2 types of attention functions corresponding



1) Channel Attention Module

So, what is channel attention and why do we need it?

The channel attention essentially provides a weight for each channel and thus enhances those particular channels which are most contributing towards learning and thus boosts the overall model performance.

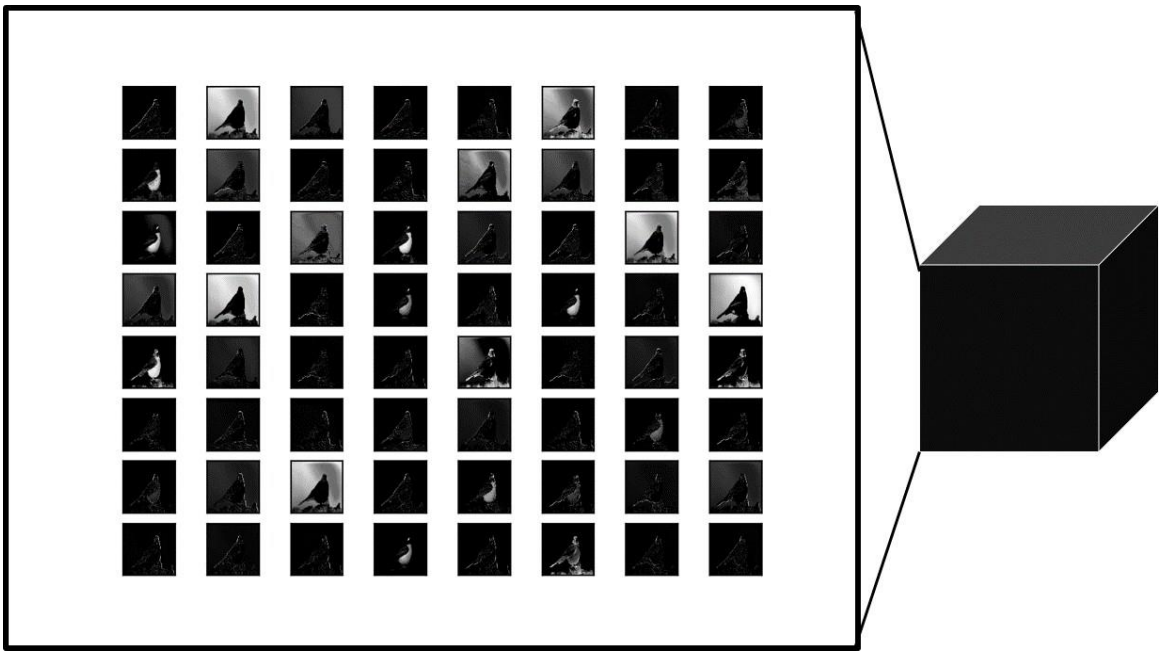
> In layman terms, channel attention says which feature map is important for learning and enhances, or as the authors say, "refines" it. Meanwhile, the spatial attention conveys what within the feature map is essential to learn. Combining both robustly enhances the Feature Maps and thus justifies the significant improvement in model performance.

> **Code for Attention module**

```
def attention_block(input, input_channels=None, output_channels=None, encoder_depth=1):
    """
    attention block
    https://arxiv.org/abs/1704.06904
    """

    p = 1
    t = 2
    r = 1

    if input_channels is None:
        input_channels = input.get_shape()[-1]
        print("input shape",input.get_shape(),input_channels)
    if output_channels is None:
        output_channels = input_channels
        print("output shape",output_channels)

    # First Residual Block
    for i in range(p):
        input = residual_block(input)
        print('p',input)

    # Trunk Branch
    output_trunk = input
    for i in range(t):
        output_trunk = residual_block(output_trunk)
        print('t',output_trunk)

    # Soft Mask Branch

    ## encoder
    ### first down sampling
    output_soft_mask = MaxPool2D(padding='same')(input)  # 32x32
    for i in range(r):
        output_soft_mask = residual_block(output_soft_mask)
        print('r',output_soft_mask)

    skip_connections = []

    for i in range(encoder_depth - 1):

        ## skip connections
        output_skip_connection = residual_block(output_soft_mask)
        skip_connections.append(output_skip_connection)
        print ('skip shape:', output_skip_connection.get_shape())

        ## down sampling
        output_soft_mask = MaxPool2D(padding='same')(output_soft_mask)
        for _ in range(r):
            output_soft_mask = residual_block(output_soft_mask)

            ## decoder
    skip_connections = list(reversed(skip_connections))
    print('skip_connections',skip_connections)
    for i in range(encoder_depth - 1):
        ## upsampling
        for _ in range(r):
            output_soft_mask = residual_block(output_soft_mask)
        output_soft_mask = UpSampling2D()(output_soft_mask)
        ## skip connections
        output_soft_mask = Add()([output_soft_mask, skip_connections[i]])

    ### last upsampling
    for i in range(r):
        output_soft_mask = residual_block(output_soft_mask)
    output_soft_mask = UpSampling2D()(output_soft_mask)

    ## Output
    output_soft_mask = Conv2D(input_channels, (1, 1))(output_soft_mask)
    output_soft_mask = Conv2D(input_channels, (1, 1))(output_soft_mask)
    output_soft_mask = Activation('sigmoid')(output_soft_mask)

    # Attention: (1 + output_soft_mask) * output_trunk
    output = Lambda(lambda x: x + 1)(output_soft_mask)
    output = Multiply()([output, output_trunk])  #

    # Last Residual Block
    for i in range(p):
        output = residual_block(output)

    return output
```

2) Spatial Attention Module

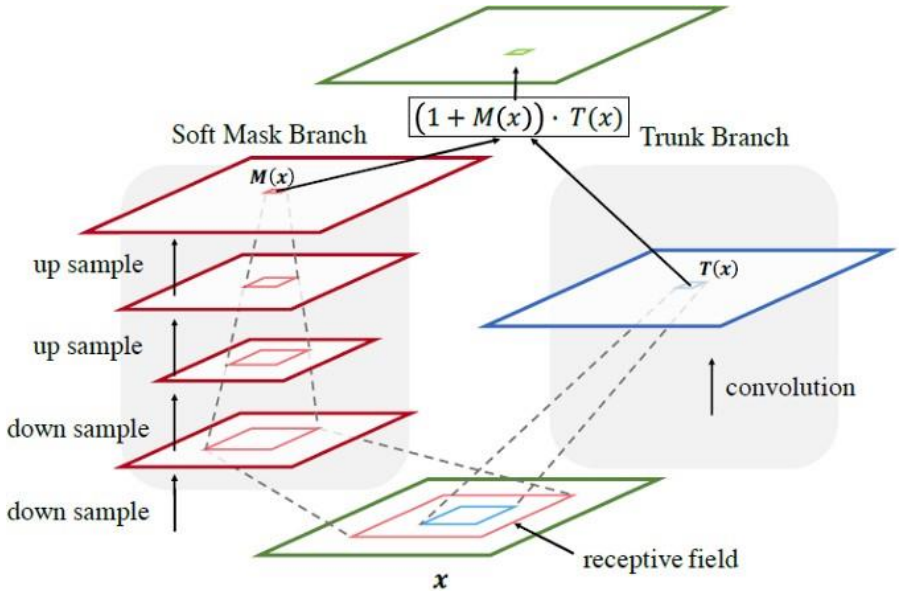
Spatial attention represents the attention mechanism/attention mask on the feature map, or a single cross-sectional slice of the tensor. For instance, in the image below the object of interest is a bird, thus the Spatial Attention will generate a mask which will enhance the features that define that bird. By thus refining the feature maps using Spatial Attention, we are enhancing the input to the subsequent convolutional layers which thus improves the performance of the model.



In subsequent convolution layers spatial attention module will enhance the features that define the bird.

SoftMax Branch

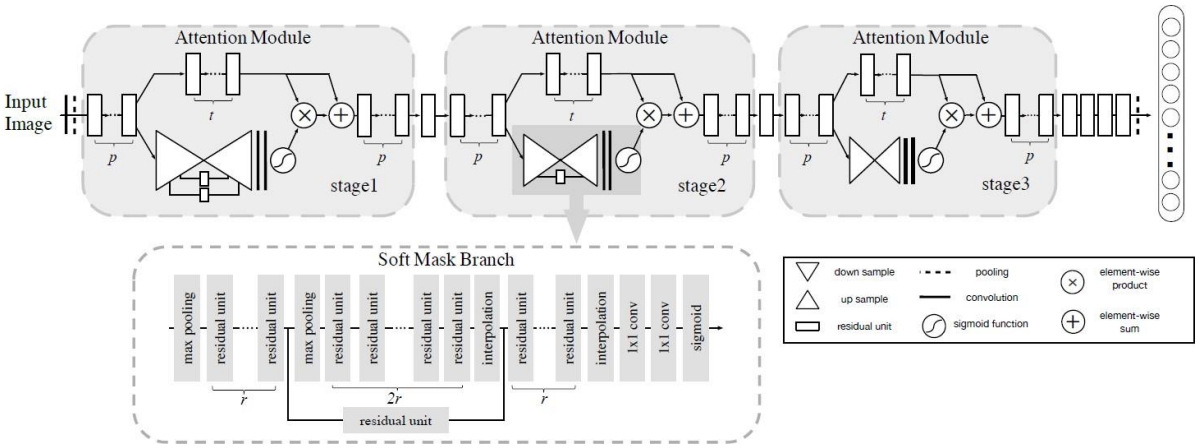
In order to learn about the model architecture, we need to first know about SoftMax Branch as this plays an important role in our architecture.



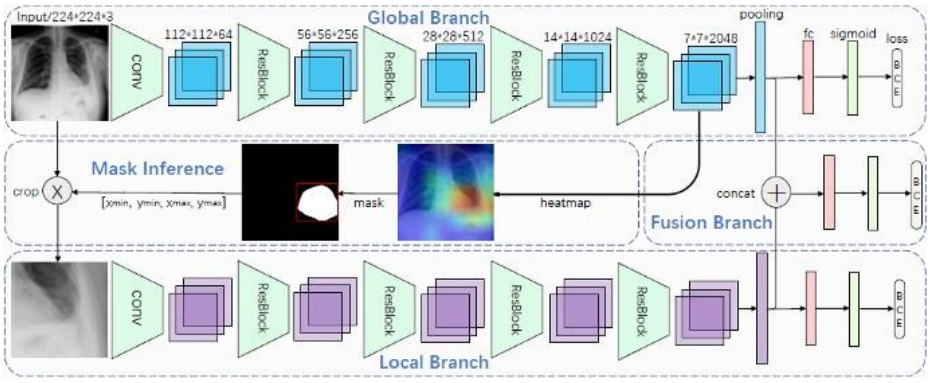
- A **bottom-up top-down fully convolutional structure** is used.
- **Max pooling are performed several times to increase the receptive field** (The receptive field is defined as the region in the input space that a particular CNN's feature is looking at (i.e., be affected by)) **rapidly** after a small number of Residual Units.
- Then, the global information is then expanded by a symmetrical top-down architecture to guide input features in each position.
- A linear interpolation up samples the output after some Residual Units.
- The number of bilinear interpolations is the same as max pooling to keep the output size the same as the input feature map.
- Then a sigmoid layer normalizes the output after two 1*1 convolution.

Model Architecture

Let me give you an overview of the architecture and then let's dive into the main part (Code).



Model Architecture

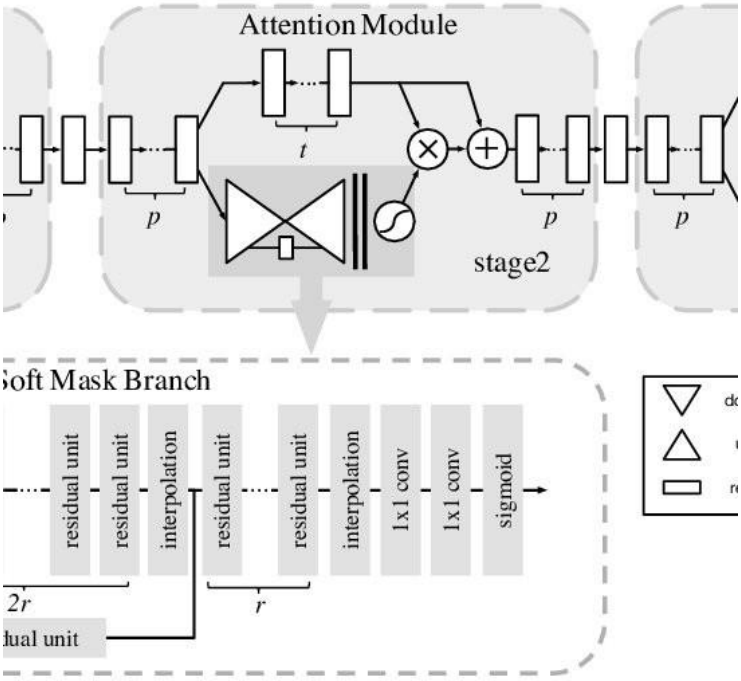


Different types of visualization in model architecture: Image credits

Residual Attention Network (RAN) is a convolutional neural network that incorporates both attention mechanism and residual units, a component that utilizes skip-connections to jump over 2-3 layers with nonlinearities (e.g., ReLU in CNNs) and batch normalizations. Its prime feature is the attention module.

Now the architecture of the attention module

The RAN is built by stacking Attention Modules, which generate attention-aware features that adaptively change as layers move deeper into the network.



Attention Module

The hyper-parameter p denotes the number of pre-processing Residual Units before splitting into trunk branch and mask branch. t denotes the number of Residual Units in trunk branch. r denotes the number of Residual Units between adjacent pooling layer in the mask branch. In our experiments, we use the following hyper-parameters setting:
{p = 1, t = 2, r = 1}. The number of channels in the soft mask Residual Unit and corresponding trunk branches is the same.

▼ The composition of the attention module includes two branches

1 The Trunk branch

2 The Mask branch

1) Trunk **Branch** performs feature processing with Residual Units

2) Mask **Branch** uses bottom-up top-down structure softly weight output features with the goal of improving trunk branch features

- **Bottom-Up Step**: collects global information of the whole image by **down sampling** (i.e., max pooling) the image
- **Top-Down Step**: combines global information with original feature maps by **up sampling** (i.e., interpolation) to keep the output size the same as the input feature map

Once the actions are completed, the features extracted from the respective branches are combined together using the team's novel Attention Residual Learning formula. This is used to train very deep Residual Attention Networks so that it can be easily scaled up to hundreds of layers without a drop in performance. Thus, increasing Attention Modules leads to consistent performance improvement, as different types of attention are captured extensively.

Previous Work Done

The CIFAR-10 and CIFAR-100 datasets consist of 60, 000 32 × 32 color images of 10 and 100 classes respectively, with 50, 000 training images and 10, 000 test images. The broadly applied state-of-the-art network structure ResNet is used as baseline method. To conduct fair comparison.

Network	params $\times 10^6$	CIFAR-10	CIFAR-100
ResNet-164 [11]	1.7	5.46	24.33
ResNet-1001 [11]	10.3	4.64	22.71
WRN-16-8 [39]	11.0	4.81	22.07
WRN-28-10 [39]	36.5	4.17	20.50
Attention-92	1.9	4.99	21.71
Attention-236	5.1	4.14	21.16
Attention-452†	8.6	**3.90**	**20.45**

Here we can see the test error of the models on CIFAR□10 and CIFAR□100 dataset and we can see here that Attention-452 model has outperformed all.

Problem Statement

As the second-largest provider of carbohydrates in Africa, cassava is a key food security crop grown by smallholder farmers because it can withstand harsh conditions. At least 80% of household farms in Sub-Saharan Africa grow this starchy root, but viral diseases are major sources of poor yields. With the help of data science, it may be possible to identify common diseases so they can betreated.

Existing methods of disease detection require farmers to solicit the help of government-funded agricultural experts to visually inspect and diagnose the plants. This suffers from being labor-intensive, low-supply and costly. As an added challenge, effective solutions for farmers must perform well under significant constraints, since African farmers may only have access to mobile-quality cameras withlow-bandwidth.

The description of the dataset is mentioned below in the "Code Explanation" part.

Code Explanation

Let's now dive into the coding part!

Step - 1

Installing Kaggle on Google colab to use the GPU in collab and downloading the dataset.

```
!pip install kaggle


from google.colab import files

files.upload()


! mkdir ~/.kaggle

! cp kaggle.json ~/.kaggle/

! chmod 600 ~/.kaggle/kaggle.json

! kaggle datasets list

!pip install --upgrade --force-reinstall --no-deps kaggle

!kaggle competitions download -c cassava-leaf-disease-classification

! mkdir Cassava


! unzip cassava-leaf-disease-classification.zip -d C
```

So let's understand this piece of code

```
!pip install kaggle
# IT is an official API for https://www.kaggle.com, accessible using a command-line #
tool implemented in Python.

from google.colab import files
files.upload()
# To upload the Json token file downloaded from kaggle.

! mkdir ~/.kaggle
# Making a directory named Kaggle.

# Move the kaggle.json file into ~/.kaggle,
# which is where the API client expects your token to be located:

!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

! chmod 600 ~/.kaggle/kaggle.json
# Chmod 600 sets permissions so that, (U) ser/owner can
# read, can write and can't execute. (G)roup can't read, can't write and can't execute #
(O)others can't read, can't write and can't.
```

```
# Creating a directory.

! unzip cassava-leaf-disease-classification.zip -d C #
Unzipping the dataset into the directory.
```

Step - 2

Data Division

Dataset was borrowed from ~~-~~Kaggle.

The dataset into 3 files —

1 **Train.csv** - It contains all the image_id and labels of the image present

train.csv

1 to 10 of 2

image_id	label
1000015157.jpg	0
1000201771.jpg	3
100042118.jpg	1
1000723321.jpg	1
1000812911.jpg	3
1000837476.jpg	3
1000910826.jpg	2
1001320321.jpg	0
1001723730.jpg	4
1001742395.jpg	3

Show 10 per page

1 2 10 100 1000 2000 2

2 **Train images** - It contains a total of **21,400 images** with ids.

Visualization of the data

3 **label_num_to_disease_map.json** - It contains all the mapping between each disease code and disease name.

Step - 3

Data Pre-Processing

I'll explain the code that is bigger and somewhat confusing and skip the easy ones (I'll explain in the main code itself).

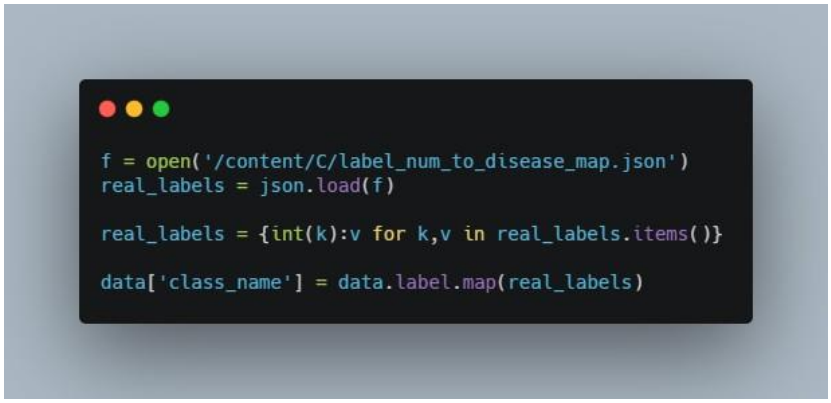
1 Checking the frequencies of all the labels in the dataset.

```
#%% IMPORTING DATA

# Importing train.csv

data = pd.read_csv('/content/C/train.csv')
print(Counter(data['label'])) # Checking the frequencies of the labels

# Output - Counter({3: 13158, 4: 2577, 2: 2386, 1: 2189, 0: 1087})
```

2 Creating a new column in the Train.csv folder by the name 'class_name' and it contains names of all the diseases corresponding to their labels.

	image_id	label	class_name
21387	997651546.jpg	3	Cassava Mosaic Disease (CMD)
21388	997857988.jpg	3	Cassava Mosaic Disease (CMD)
21389	997910101.jpg	2	Cassava Green Mottle (CGM)
21390	997973414.jpg	1	Cassava Brown Streak Disease (CBSD)
21391	998910982.jpg	1	Cassava Brown Streak Disease (CBSD)
21392	999068805.jpg	3	Cassava Mosaic Disease (CMD)
21393	999329392.jpg	3	Cassava Mosaic Disease (CMD)
21394	999474432.jpg	1	Cassava Brown Streak Disease (CBSD)
21395	999616605.jpg	4	Healthy
21396	999998473.jpg	4	Healthy

How did we do it?



```
f = open('/content/C/label_num_to_disease_map.json')
real_labels = json.load(f)

real_labels = {int(k):v for k,v in real_labels.items()}

data['class_name'] = data.label.map(real_labels)
```

So, first, open the JSON file(**label_num_to_disease_map.json)** and load it.

```
f = open('/content/C/label_num_to_disease_map.json')
real_labels = json.load(f) #Opening and Loading the JSON file.
```

Then, as we can see in the JSON file that the key values have a **string** data type and in the **'Train.csv'** they are having "int" data type. We need to convert the string data type to integer or else it will give us an error.

```
real_labels = {int(k):v for k,v in real_labels.items()} # Converting str to int.
#Before - {'0': 'Cassava Bacterial Blight (CBB)', '1': 'Cassava Brown Streak Disease (CBSD)', '2': 'Cassava Green Mottle (CGM)', '3': #After - {0: 'Cassava Bacterial Blight (CBB)', 1: 'Cassava Brown Streak Disease (CBSD)', 2: 'Cassava Green Mottle (CGM)', 3: 'Cassava
```

Then, after converting it to int then we create a new column as 'class_name' and we just need to use the map function and it will take care of the rest.

```
data['class_name'] = data.label.map(real_labels) # Mapping
```

3 Splitting the dataset into **80 20%** and using stratify on **class_name** so as to have the same number of labels in training and testing dataset.

```
from sklearn.model_selection import train_test_split

train,val = train_test_split(data, test_size = 0.2, random_state = 42, stratify = data['class_name'])
```

Step - 4

Data Augmentation

Data augumentation on training data.

```
from keras.preprocessing.image import ImageDataGenerator

datagen_train = ImageDataGenerator(
                    rescale = 1./255,
                    rotation_range = 40,
                    width_shift_range = 0.2,
                    height_shift_range = 0.2,
                    shear_range = 0.2,
                    zoom_range = 0.2,
                    horizontal_flip = True,
                    vertical_flip = True,
                    #validation_split = 0.73,
                    fill_mode = 'nearest')
```

So let's try to understand this part of the code

Images after Rotation

In spite of all the data availability, fetching the right type of data which matches the exact use-case of our experiment is a daunting task. Moreover, the data has to have good diversity as the object of interest needs to be present in varying sizes, lighting conditions and poses if we desire that our network generalizes well during the testing (or deployment) phase. To overcome this problem of limited quantity and limited diversity of data, we generate(manufacture) our own data with the existing data which we have. This methodology of generating our own data is known as data augmentation.

Data augmentation on testing data.

```
datagen_val = ImageDataGenerator(rescale = 1./255)
```

Let's understand this piece of code

The goal of data augmentation is to generalize the model and make it learn more orientation of the images, such that the during testing the model is able to apprehend the test data well. So, it is well practiced to use augmentation technique only for training sets.

Step - 5

Now comes the main part - Model Architecture

code credits : Github

The architecture of this model is build based on this research paper.

Layers	Output Size	Kernel Size
Convolution 2D	112×112	(5×5), p=same
MaxPool 2D	56×56	(2×2), 2
Residual Block	56×56	$\begin{pmatrix} 1\times1 & 32 \\ 3\times3 & 32 \\ 1\times1 & 128 \end{pmatrix}$
Attention Block	56×56	Attention×1
Residual Block	28×28	$\begin{pmatrix} 1\times1 & 128 \\ 3\times3 & 128 \\ 1\times1 & 256 \end{pmatrix}$
Attention Block	28×28	Attention×1
Residual Block	14×14	$\begin{pmatrix} 1\times1 & 256 \\ 3\times3 & 256 \\ 1\times1 & 512 \end{pmatrix}$
Attention Block	14×14	Attention×1
Residual Block	7×7	$\begin{pmatrix} 1\times1 & 512 \\ 3\times3 & 512 \\ 1\times1 & 1024 \end{pmatrix}$
Residual Block	7×7	$\begin{pmatrix} 1\times1 & 1024 \\ 3\times3 & 1024 \\ 1\times1 & 1024 \end{pmatrix}$
Residual Block	7×7	$\begin{pmatrix} 1\times1 & 1024 \\ 3\times3 & 1024 \\ 1\times1 & 1024 \end{pmatrix}$
AvgPooling 2D	1×1	(7×7)
FC, Softmax	2	
Depth	115	

Code for this architecture

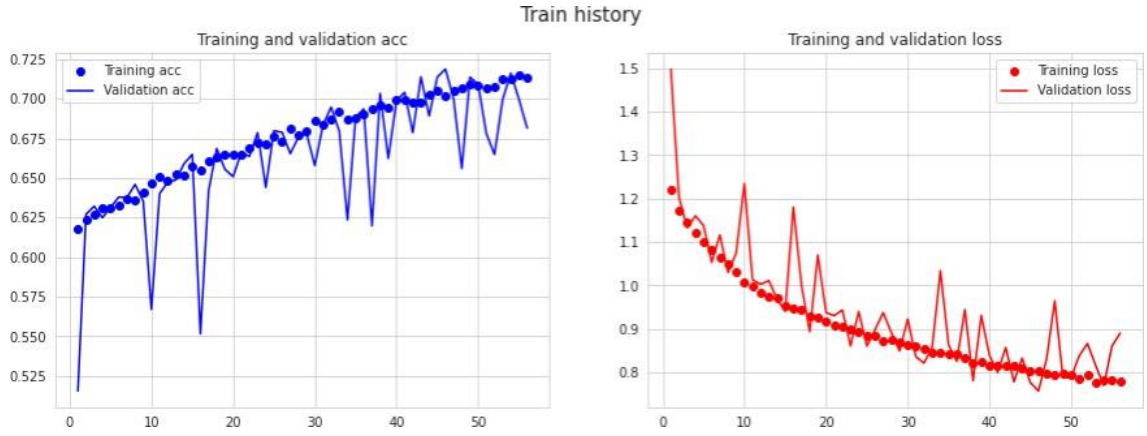
```python
def AttentionResNet56(shape=(224, 224, 3), n_channels=64, n_classes=100,
                      dropout=0, regularization=0.01):
    """
    Attention-56 ResNet
    https://arxiv.org/abs/1704.06904
    """

    regularizer = l2(regularization)

    input_ = Input(shape=shape)
    x = Conv2D(n_channels, (7, 7), strides=(2, 2), padding='same')(input_) # 112x112
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = MaxPool2D(pool_size=(3, 3), strides=(2, 2), padding='same')(x)  # 56x56

    x = residual_block(x, output_channels=n_channels * 4)  # 56x56
    x = attention_block(x, encoder_depth=3)  # bottleneck 7x7

    x = residual_block(x, output_channels=n_channels * 8, stride=2)  # 28x28
    x = attention_block(x, encoder_depth=2)  # bottleneck 7x7

    x = residual_block(x, output_channels=n_channels * 16, stride=2)  # 14x14
    x = attention_block(x, encoder_depth=1)  # bottleneck 7x7

    x = residual_block(x, output_channels=n_channels * 32, stride=2)  # 7x7
    x = residual_block(x, output_channels=n_channels * 32)
    x = residual_block(x, output_channels=n_channels * 32)

    pool_size = (x.get_shape()[1].value, x.get_shape()[2].value)
    x = AveragePooling2D(pool_size=pool_size, strides=(1, 1))(x)
    x = Flatten()(x)
    if dropout:
        x = Dropout(dropout)(x)
    output = Dense(n_classes, kernel_regularizer=regularizer, activation='softmax')(x)

    model = Model(input_, output)
    return model
```

Conclusion

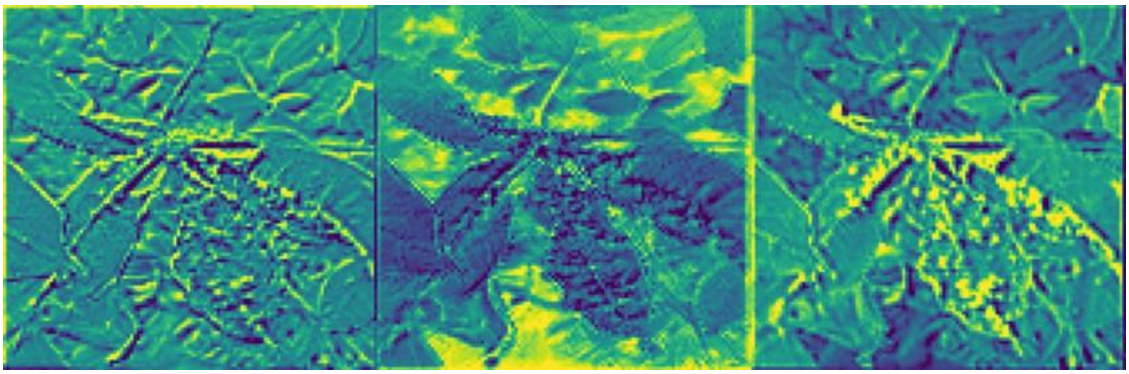
We propose a Residual Attention Network which stacks multiple Attention Modules. The benefits of our network are in two folds: it can capture mixed attention and is an extensible convolutional neural network. The **first benefit** lies in that different Attention Modules capture different types
of attention to guide feature learning. Our experiments on the forms of activation function also validate this point: free form mixed attention will have better performance than
constrained (including single) attention. The **second benefit** comes from encoding top-down attention mechanism into bottom-up top-down feedforward convolutional structure in each Attention Module.

Visualizations of the output till now

Results of the model after training for 56 epochs.



Visualization of attention layer.



Future Work

As we can see that in the output that along with **accuracy** the **validation accuracy** is also constantly **increasing** and along with **loss** the **validation loss** is also constantly **decreasing.**

- So, to improve our model we can try our model with different augmentations.
- We could train the models using cross validation techniques.
- We could use ensemble modelling also to increase our accuracy.

Visual Representation of the model architecture