

MODERN AI PRACTICAL



A comprehensive course covering state-of-the-art AI applications such as Emotion AI, Creative AI, Explainable AI, Business and HealthCare AI, and many more.

By Dr. Ryan Ahmed, Ph.D., MBA
SuperDataScience Team



1. EMOTION AI



A. CONCEPT

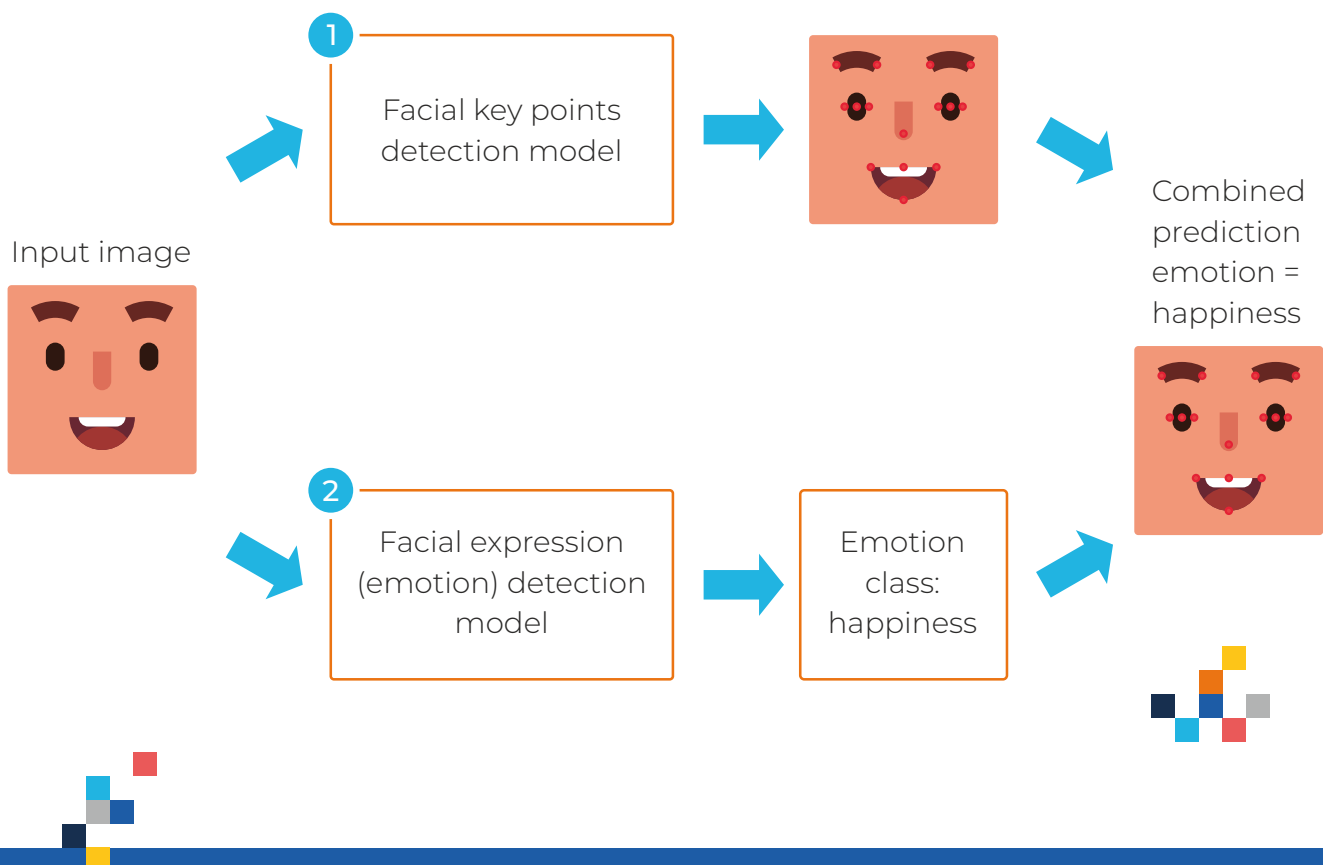
- Artificial Emotional Intelligence or Emotion AI is a branch of AI that allow computers to understand human non-verbal cues such as body language and facial expressions.
- Affectiva offers cutting edge emotion AI tech:
<https://www.affectiva.com/>

B. PROJECT OVERVIEW

- The aim of this project is to classify people's emotions based on their face images.
- In this case study, we will assume that you work as an AI/ML consultant. You have been hired by a Startup in San Diego to build, train and deploy a system that automatically monitors people emotions and expressions.

C. NETWORKS PIPELINE

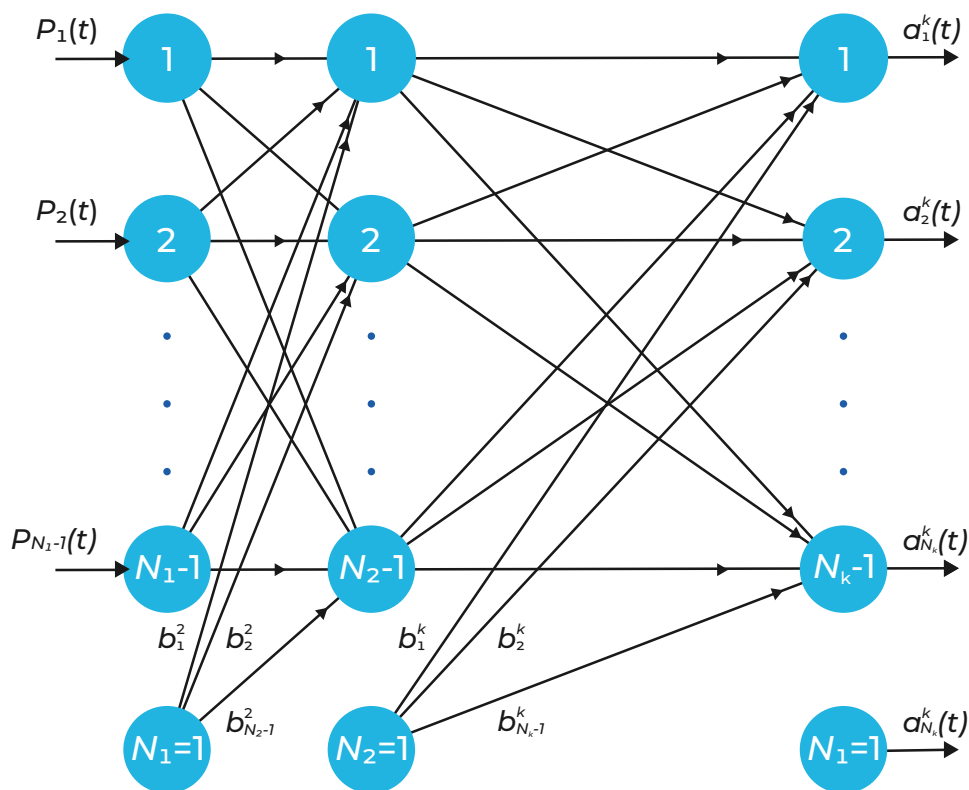
- We pass our image through two different models, one is a key-point detection model and the other is an expression detection-based model.
- Finally, we combine the output of these two models to get the final result.



D. DEEP LEARNING AND COMPUTER VISION

1. ARTIFICIAL NEURAL NETWORKS CONCEPT

- Artificial neural networks are information processing models inspired by the human brain. Anns are built in a layered fashion where inputs are propagated starting from the input layer through the hidden layers and finally to the output.



Networks training is performed by optimizing the matrix of weights outlined below:

$$\begin{bmatrix} W_{11} & W_{12} & \dots & W_{1,N_1} \\ W_{21} & W_{22} & \dots & W_{2,N_1} \\ \vdots & \vdots & \ddots & \vdots \\ W_{m-1,1} & W_{m-1,2} & \dots & W_{m-1,N_1} \\ W_{m,1} & W_{m,2} & \dots & W_{m,N_1} \end{bmatrix}$$

2. BUILD AN ANNS USING KERAS

```
>> import tensorflow.keras
>> from keras.models import Sequential
>> from keras.layers import Dense
>> from sklearn.preprocessing import MinMaxScaler

>> model = Sequential()
>> model.add(Dense(25, input_dim=5, activation='relu'))
>> model.add(Dense(25, activation='relu'))
>> model.add(Dense(1, activation='linear'))
>> model.summary()
```

3. TRAIN AN ANN USING KERAS

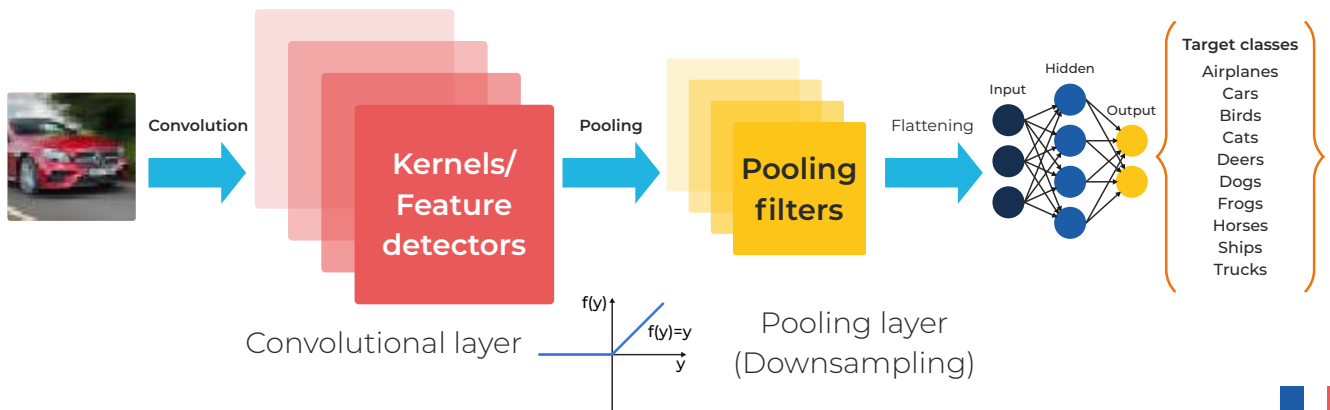
```
>> model.compile(optimizer='adam', loss='mean_squared_error')
>> epochs_hist = model.fit(X_train, y_train, epochs=20,
batch_size=25, validation_split=0.2)
```

4. EVALUATE THE TRAINED ANN MODEL

```
>> X_Testing = np.array([[input #1, input #2, input #3,..., input
#n]])
>> y_predict = model.predict(X_Testing)
```

5. HOW TO BUILD A CONVOLUTIONAL NEURAL NETWORK (CNN)

- CNNs is a type of deep neural networks that are commonly used for image classification.
- CNNs are formed of (1) Convolutional Layers (Kernels and feature detectors), (2) Activation Functions (RELU), (3) Pooling Layers (Max Pooling or Average Pooling), and (4) Fully Connected Layers (Multi-layer Perceptron Network).



6. BUILD A CNN USING KERAS

```
>> from keras.models import Sequential
>> from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D,
Dense, Flatten, Dropout
>> from keras.optimizers import Adam >> from keras.callbacks import
TensorBoard

>> cnn_model = Sequential()

>> cnn_model.add(Conv2D(filters = 32, kernel_size=(3,3), activation =
'relu', input_shape = (32,32,3)))
>> cnn_model.add(Conv2D(filters = 32, kernel_size=(3,3), activation =
'relu'))
>> cnn_model.add(MaxPooling2D(2,2))
>> cnn_model.add(Dropout(0.3))

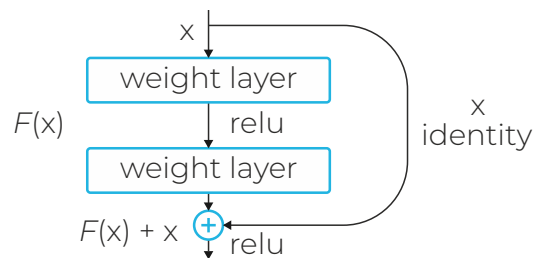
>> cnn_model.add(Flatten())

>> cnn_model.add(Dense(units = 512, activation = 'relu'))
>> cnn_model.add(Dense(units = 10, activation = 'softmax'))
```

7. WHAT IS RESIDUAL NETWORK

- As CNNs grow deeper, vanishing gradient tend to occur which negatively impact network performance.
- Vanishing gradient problem occurs when the gradient is backpropagated to earlier layers which results in a very small gradient.

- Residual Neural Network includes “skip connection” feature which enables training of 152 layers without vanishing gradient issues.
- Resnet works by adding “identity mappings” on top of CNN.
- ImageNet contains 11 million images and 11,000 categories.
- ImageNet is used to train ResNet deep network.




8. KERAS IMPLEMENTATION OF RESNET

```

>>def res_block(X, filter, stage)
>>X_copy = X >>f1 , f2, f3 = filter
>>X = Conv2D(f1, (1,1),strides = (1,1), kernel_initializer=
glorot_uniform(seed = 0))(X)
>>X = MaxPool2D((2,2))(X)
>>X = BatchNormalization(axis =3)(X)
>>X = Activation('relu')(X)
>>X = Conv2D(f2, kernel_size = (3,3), strides =(1,1), padding =
'same', kernel_initializer= glorot_uniform(seed = 0))(X)
>>X = BatchNormalization(axis =3,)(X)
>>X = Activation('relu')(X)
>>X = Conv2D(f3, kernel_size = (1,1), strides =(1,1) ,
kernel_initializer= glorot_uniform(seed = 0))(X)
>>X = BatchNormalization(axis =3)(X)
# Short pat
>>X_copy = Conv2D(f3, kernel_size = (1,1), strides =(1,1),
kernel_initializer= glorot_uniform(seed = 0))(X_copy)
>>X_copy = MaxPool2D((2,2))(X_copy)
>>X_copy = BatchNormalization(axis =3)(X_copy)
# ADD
>>X = Add()([X,X_copy])
>>X = Activation('relu')(X)
# Identity Block 1
>>X_copy = X
# Main Path


```



```

>>X = Conv2D(f1, (1,1),strides = (1,1), kernel_initializer=
glorot_uniform(seed = 0))(X)
>>X = BatchNormalization(axis =3)(X)
>>X = Activation('relu')(X)
>>X = Conv2D(f2, kernel_size = (3,3), strides =(1,1), padding =
'same', kernel_initializer= glorot_uniform(seed = 0))(X)
>>X = BatchNormalization(axis =3)(X)
>>X = Activation('relu')(X)
>>X = Conv2D(f3, kernel_size = (1,1), strides =(1,1),
kernel_initializer= glorot_uniform(seed = 0))(X)
>>X = BatchNormalization(axis =3)(X)
# ADD
>>X = Add()([X,X_copy])
>>X = Activation('relu')(X)
# Identity Block 2
>>X_copy = X
# Main Path
>>X = Conv2D(f1, (1,1),strides = (1,1) kernel_initializer=
glorot_uniform(seed = 0))(X)
>>X = BatchNormalization(axis =3)(X)
>>X = Activation('relu')(X)
>>X = Conv2D(f2, kernel_size = (3,3), strides =(1,1), padding =
'same', kernel_initializer= glorot_uniform(seed = 0))(X)
>>X = BatchNormalization(axis =3)(X)
>>X = Activation('relu')(X)
>>X = Conv2D(f3, kernel_size = (1,1), strides =(1,1),
kernel_initializer= glorot_uniform(seed = 0))(X)
>>X = BatchNormalization(axis =3)(X)
# ADD
>>X = Add()([X,X_copy])
>>X = Activation('relu')(X)
>>return X

```



E. TENSORFLOW SERVING

- TensorFlow Serving is a high-performance serving system for machine learning models, designed for production environments.
- With the help of TensorFlow Serving, we can easily deploy new algorithms to make predictions.
- In-order to serve the trained model using TensorFlow Serving, we need to save the model in the format that is suitable for serving using TensorFlow Serving.

- The model will have a version number and will be saved in a structured directory.
- After the model is saved, we can now use TensorFlow Serving to start making inference requests using a specific version of our trained model "servable".



1. SAVE THE TRAINED MODEL

```
>> import tempfile # Obtain a temporary storage directory
>> MODEL_DIR = tempfile.gettempdir()
>> version = 1 # specify the model version, choose #1 for now

# Let's join the temp model directory with our chosen version number
>> export_path = os.path.join(MODEL_DIR, str(version))
>> print('export_path = {}'.format(export_path))

# Save the model using simple_save
>> if os.path.isdir(export_path):
print('\nAlready saved a model, cleaning up\n')
!rm -r {export_path}

>> tf.saved_model.simple_save(
keras.backend.get_session(),
export_path,
inputs={'input_image': model.input},
outputs={t.name:t for t in model.outputs})
```

2. ADD TENSORFLOW-MODEL-SERVER PACKAGE TO OUR LIST OF PACKAGES

```
>> !echo "deb http://storage.googleapis.com/tensorflow-serving-apt
stable tensorflow-model-server >> tensorflow-model-server-universal" |
tee /etc/apt/sources.list.d/tensorflow-serving.list && \ curl
https://storage.googleapis.com/tensorflow-serving-
apt/tensorflowserving.release.pub.gpg | apt-key add -
>> !apt update
```

3. INSTALL TENSORFLOW MODEL SERVER

```
>> !apt-get install tensorflow-model-server
```



4. RUN TENSORFLOW SERVING

- There are some important parameters:
 - `rest_api_port`: The port that you'll use for REST requests.
 - `model_name`: You'll use this in the URL of REST requests. You can choose any name.
 - `model_base_path`: This is the path to the directory where you've saved your model.
- For more information regarding REST, check this out:
<https://www.codecademy.com/articles/what-is-rest>
- REST is a revival of HTTP in which http commands have semantic meaning.

```
>> os.environ["MODEL_DIR"] = MODEL_DIR
>> %bash --bg
>> nohup tensorflow_model_server \
--rest_api_port=8501 \
--model_name=fashion_model \
--model_base_path="${MODEL_DIR}" >server.log 2>&1

>> !tail server.log
```

5. START MAKING REQUESTS IN TENSORFLOW SERVING

- In-order to make prediction using TensorFlow Serving, we need to pass the inference requests (image data) as a JSON object.

- Finally, we get the prediction from the post request made to the deployed model and then use argmax function to find the predicted class.
- Then, we use python requests library to make a post request to the deployed model, by passing in the JSON object containing inference requests (image data).

```
# Let's create a JSON object and make 3 inference requests
>> data = json.dumps({"signature_name": "serving_default",
"instances": test_images[0:10].tolist()})
>> print('Data: {} ... {}'.format(data[:50], data[len(data)-52:]))
>> !pip install -q requests
>> import requests
>> headers = {"content-type": "application/json"}
>> json_response =
requests.post('http://localhost:8501/v1/models/fashion_model:predict
', data=data, headers=headers)
>> predictions = json.loads(json_response.text)['predictions']
>> show(0, 'The model thought this was a {} (class {}), and it was
actually a {} (class {})' .format(
class_names[np.argmax(predictions[0])], test_labels[0],
class_names[np.argmax(predictions[0])], test_labels[0]))
```

2. BUSINESS AI



A. CONCEPT

- In this case study, we will assume that you work as a data scientist at a bank.
- The bank has collected extensive data about its customers such as demographics, historical payments record, and amount of bill dollar values.
- The data consists of 25 variables.

B. XGBOOST

- XGBoost or Extreme gradient boosting is the algorithm of choice for many data scientists and could be used for regression and classification tasks.

XGBoost is a supervised learning algorithm and implements gradient boosted trees algorithm.

The algorithm work by combining an ensemble of predictions from several weak models.

It is robust to many data distributions and relationships and offers many hyperparameters to tune model performance.

C. XGBOOST- BOOSTING



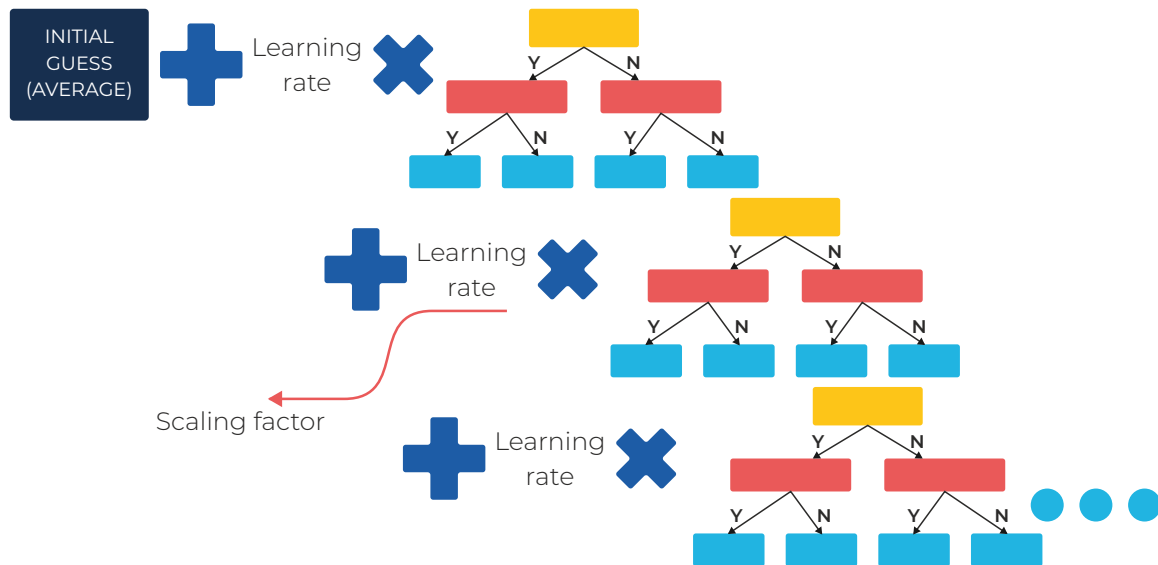
- XGBoost is an example of ensemble learning.
- Ensemble techniques such as bagging and boosting can offer an extremely powerful algorithm by combining a group of relatively weak/average ones.
- For example, you can combine several decision trees to create a powerful random forest algorithm.
- By Combining votes from a pool of experts, each will bring their own experience and background to solve the problem resulting in a better outcome.
- Boosting can reduce variance and overfitting and increase the model robustness.



D. XGBOOST- GRADIENT BOOSTING ALGORITHM

- Gradient boost works by building a tree based on the error (residuals) from the previous tree.

- Gradient boost scales the trees and then adds the predictions from the new tree to the predictions from previous trees.



E. XGBOOST- IMPLEMENTATION

```
>> import xgboost as xgb
>> model = xgb.XGBClassifier(objective='reg:squarederror',
learning_rate = 0.1, max_depth = 5, n_estimators = 100)
>> model.fit(X_train, y_train)
>> y_pred = model.predict(X_test)

>> param_grid = {'gamma': [0.5, 1, 5], 'subsample': [0.6, 0.8, 1.0],
'colsample_bytree': [0.6, 0.8, 1.0], 'max_depth': [3, 4, 5] }
>> xgb_model = XGBClassifier(learning_rate=0.01, n_estimators=100,
objective='binary:logistic')
>> from sklearn.model_selection import GridSearchCV
>> grid = GridSearchCV(xgb_model, param_grid, refit = True, verbose=4)
>> grid.fit(X_train, y_train)
```

F. XGBOOST IN AWS SAGEMAKER

- Gradient boosting uses tabular data for inputs/outputs:
 - Rows represent observations.
 - One column represents the output or target label.

- The rest of the columns represent the inputs (features).
- Amazon SageMaker implementation of XGBoost supports the following file format for training and inference:
 - CSV.
 - libsvm.
- Xgboost does not support protobuf format (note: this is unique compared to other Amazon SageMaker algorithms, which use the protobuf training input format).
- XGBoost currently only trains using CPUs.
- Xgboost is memory intensive algorithm so it does not require much compute.
- M4: General-purpose compute instance is recommended.
- There is over 40 hyperparameters to tune Xgboost algorithm with AWS SageMaker.
- Here're the tree most important ones:
 - Max_depth (0 – inf): is critical to ensure that you have the right balance between bias and variance. If the max_depth is set too small, you will underfit the training data. If you increase the max_depth, the model will become more complex and will overfit the training data. Default value is 6.
 - Gamma (0 – inf): Minimum loss reduction needed to add more partitions to the tree.
 - Eta (0 – 1): step size shrinkage used in update to prevents overfitting and make the boosting process more conservative.

After each boosting step, you can directly get the weights of new features, and eta shrinks the feature weights.

- Alpha: L1 regularization term on weights. Regularization term to avoid overfitting. The higher the gamma the higher the regularization. If gamma is set to zero, no regularization is put in place.
- Lambda: L2 regularization.

3. EXPLAINABLE AI



A. CONCEPT

- Explainable AI helps answer the following key question: “why did AI made this predictions?”
- AI models are considered black boxes that take in input and return an output.
- Extensive research is being done in the area of AI explainability as it

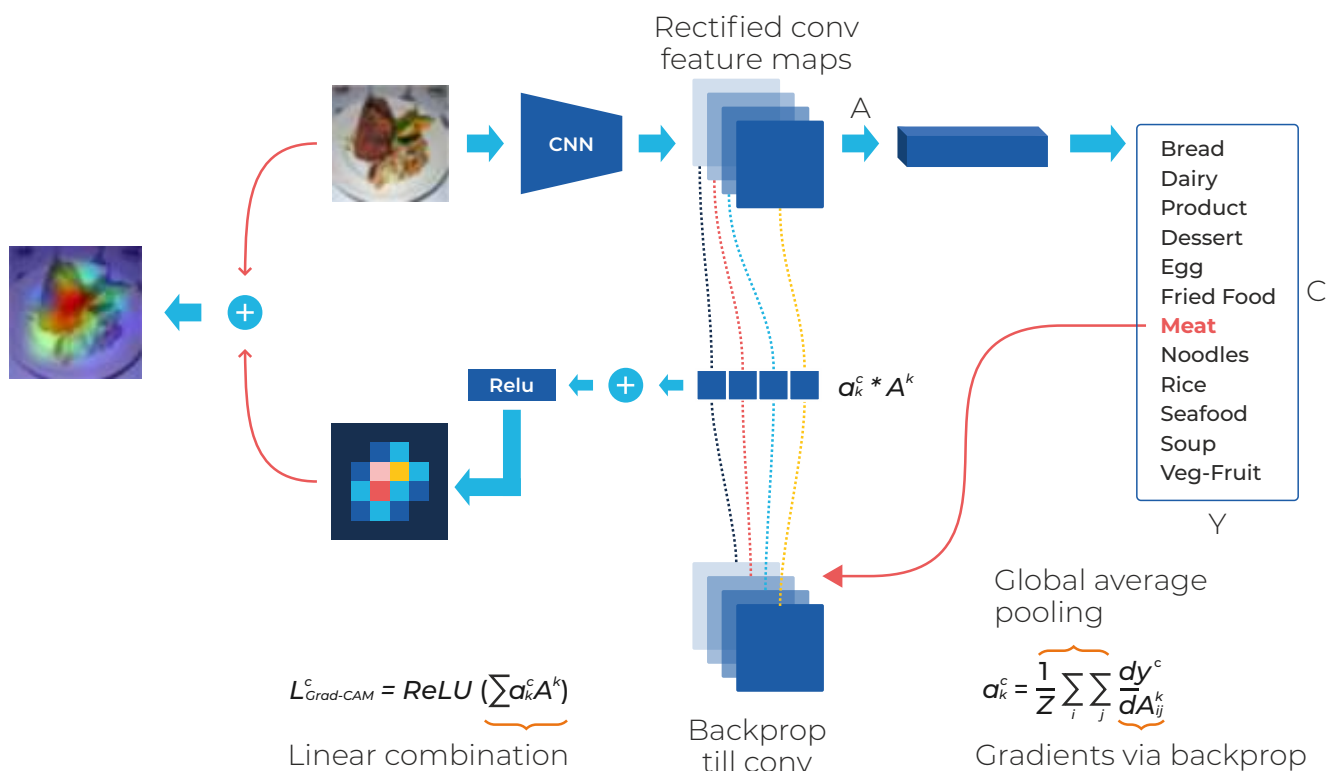


necessary to understand why this model has made this decision.

- For example, in AI applications in healthcare, doctors must understand and validate that the AI is making prediction based on right factors rather than based on irrelevant factors or noises.

B. GRAD-CAM VISUALIZATION

- Gradient-Weighted Class Activation Mapping (Grad-CAM) helps visualize the regions of the input that contributed towards making prediction by the model.
- It does so by using the class-specific gradient information flowing into the final convolutional layer of CNN to localize the important regions in the image that resulted in predicting that particular class.



C. GRAD-CAM VISUALIZATION

- To visualize the activation maps, we pass the image through the model to make the prediction.
- Use `argmax` to find the index corresponding to the maximum value in the prediction, which gives the predicted class. Now, we take the predicted value for that class by the model.
- Then, calculate the gradient that is used to arrive at that value with respect to feature map activations \mathbf{A}^k of the convolution layer.
- We are using `tensorflow.GradientTape()` to get the value of gradients.
- Since, we want to enhance the filter values that resulted in this prediction, we multiply the filter values obtained using `tensorflow.GradientTape()` with the filter values in the last convolutional layer.
- This would enhance the filter values that contributed towards making this prediction and lower the filter values that didn't contribute towards the result.
- Then, we perform weighted combination of activation maps and follow it by a ReLU to obtain the heatmap.



4. HEALTHCARE AI



A. CONCEPT

- Artificial Intelligence is revolutionizing Healthcare in many areas such as:
 - Disease Diagnosis with medical imaging.
 - Surgical Robots.
 - Maximizing Hospital Efficiency.
- AI healthcare market is expected to reach \$45.2 billion USD by 2026 from the current valuation of \$4.9 billion USD.
- Deep learning has been proven to be superior in detecting diseases from X-rays, MRI scans and CT scans which could significantly improve the speed and accuracy of diagnosis.

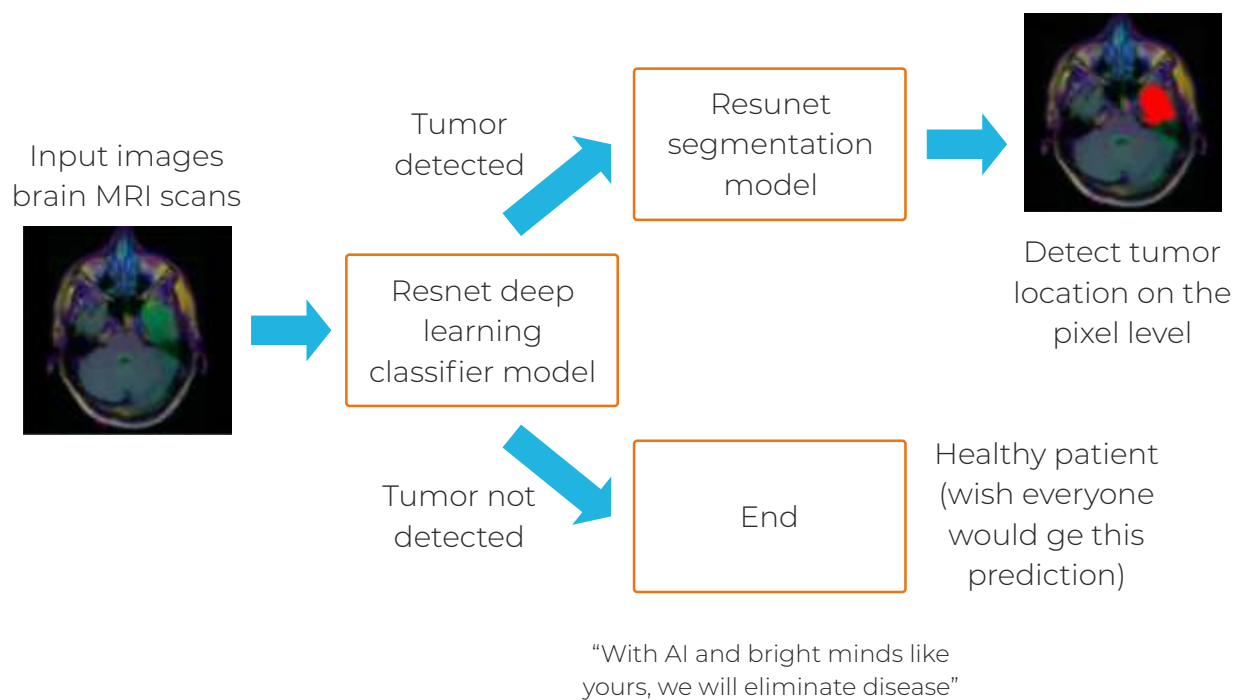
- Our goal is to improve the speed and accuracy of detecting and localizing brain tumors based on MRI scans.
- This would drastically reduce the cost of cancer diagnosis and help in early diagnosis of tumors which would essentially be a life saver.

B. IMAGE SEGMENTATION

- The goal of image segmentation is to understand and extract information from images at the pixel-level.
- Image Segmentation can be used for object recognition and localization which offers tremendous value in many applications such as medical imaging and self-driving cars etc.
- The goal of image segmentation is to train a neural network to produce pixel-wise mask of the image.
- You will use ResUNet architecture to solve the current task.
- In case of Unet, we convert (encode) the image into a vector followed by up sampling (decode) it back again into an image.
- In case of Unet, the input and output have the same size so the size of the image is preserved.



C. LAYERED DEEP LEARNING PIPELINE



D. RESUNET

- ResUNet architecture combines UNet backbone architecture with residual blocks to overcome the vanishing gradients problems present in deep architectures.
- Unet architecture is based on Fully Convolutional Networks and modified in a way that it performs well on segmentation tasks.
- Resunet consists of three parts:
 1. Encoder or contracting path.
 2. Bottleneck.
 3. Decoder or expansive path.
- 1. Encoder or contracting path consist of 4 blocks:
 - First block consists of 3x3 convolution layer + Relu + BatchNormalization.

- Remaining three blocks consist of Res-blocks followed by Maxpooling 2x2.

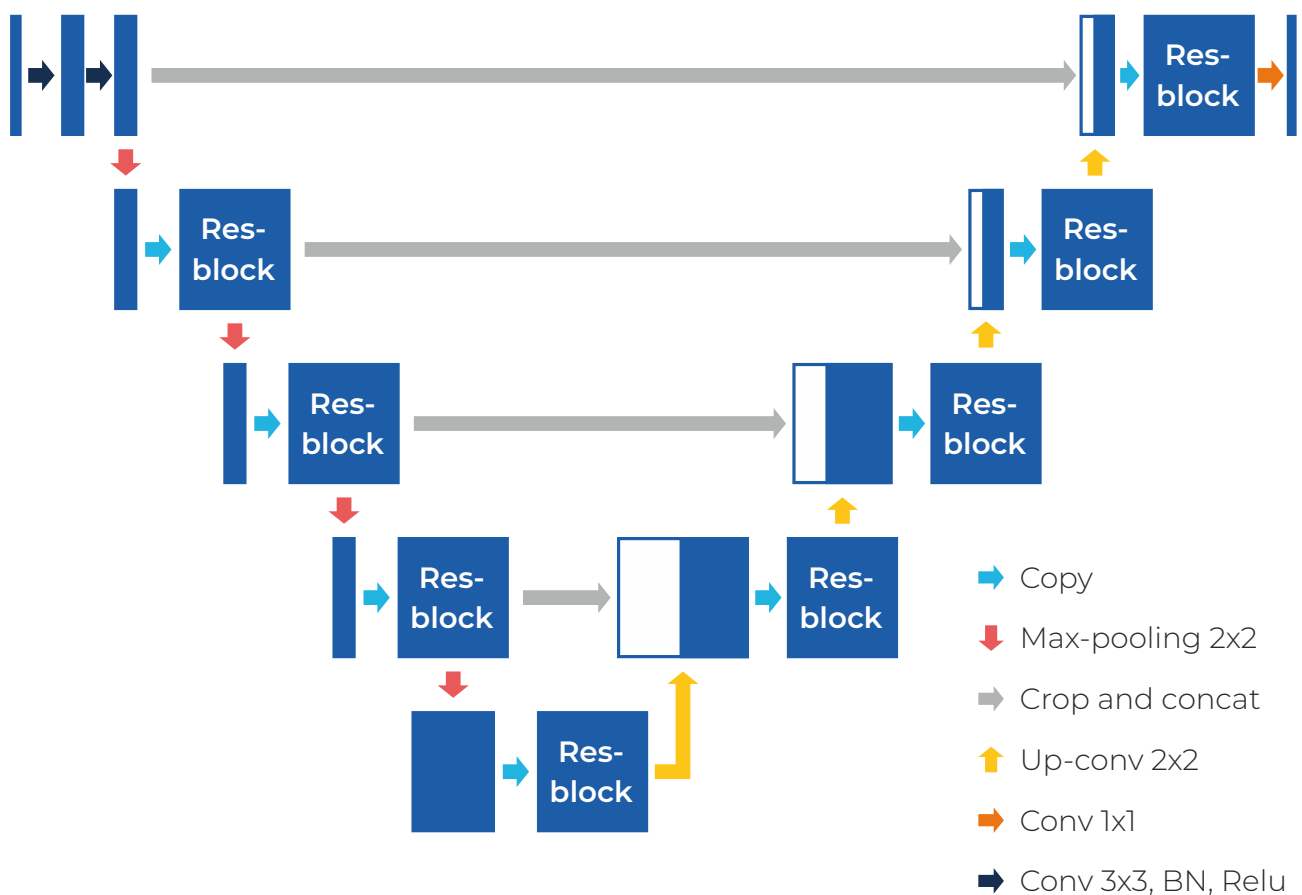
2. Bottleneck:

- It is in-between the contracting and expanding path.
- It consist of Res-block followed by up sampling conv layer 2x2.

3. Expanding or Decoder path consist of 4 blocks:

- 3 blocks following bottleneck consist of Res-blocks followed by up-sampling conv layer 2 x 2.
- Final block consist of Res-block followed by 1x1 conv layer.

E. RESUNET ARCHITECTURE





F. RESUNET IMPLEMENTATION

```
>>input_shape = (256,256,3)

# Input tensor shape
>>X_input = Input(input_shape)

# Stage 1
>>conv1_in = Conv2D(16,3,activation= 'relu', padding = 'same',
kernel_initializer = 'he_normal')(X_input)
>>conv1_in = BatchNormalization()(conv1_in)
>>conv1_in = Conv2D(16,3,activation= 'relu', padding = 'same',
kernel_initializer = 'he_normal')(conv1_in)
>>conv1_in = BatchNormalization()(conv1_in)
>>pool_1 = MaxPool2D(pool_size = (2,2))(conv1_in)

# Stage 2
>>conv2_in = resblock(pool_1, 32)
>>pool_2 = MaxPool2D(pool_size = (2,2))(conv2_in)

# Stage 3
>>conv3_in = resblock(pool_2, 64)
>>pool_3 = MaxPool2D(pool_size = (2,2))(conv3_in)

# Stage 4
>>conv4_in = resblock(pool_3, 128)
>>pool_4 = MaxPool2D(pool_size = (2,2))(conv4_in)

# Stage 5 (Bottle Neck)
>>conv5_in = resblock(pool_4, 256)

# Upscale stage 1
>>up_1 = upsample_concat(conv5_in, conv4_in)
>>up_1 = resblock(up_1, 128)

# Upscale stage 2
>>up_2 = upsample_concat(up_1, conv3_in)
>>up_2 = resblock(up_2, 64)

# Upscale stage 3
>>up_3 = upsample_concat(up_2, conv2_in)
>>up_3 = resblock(up_3, 32)

# Upscale stage 4
>>up_4 = upsample_concat(up_3, conv1_in)
>>up_4 = resblock(up_4, 16)

# Final Output
>>output = Conv2D(1, (1,1), padding = "same", activation =
"sigmoid")(up_4)

>>model_seg = Model(inputs = X_input, outputs = output )
```



G. MASK

- The goal of image segmentation is to understand the image at the pixel level. It associates each pixel with a certain class. The output produce by image segmentation model is called a “mask” of the image.
- Masks can be represented by associating pixel values with their coordinates. For example if we have a black image of shape (2,2), this can be represented as:

 $[[0, 0],$
 $[0, 0]]$

If our output mask is as follows:

 $[[255, 0],$
 $[0, 255]]$

- To represent this mask we have to first flatten the image into a 1-D array. This would result in something like $[255, 0, 0, 255]$ for mask. Then, we can use the index to create the mask. Finally we would have something like $[1, 0, 0, 1]$ as our mask.

5. MARKETING AI



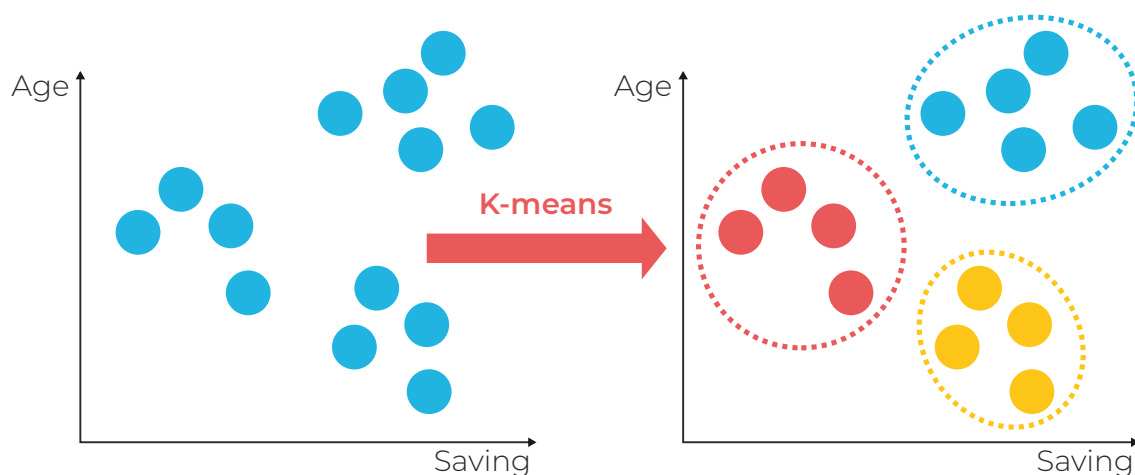
A. CONCEPT

- Marketing is crucial for the growth and sustainability of retail business.
- Marketers can help build the company's brand, engage customers, grow revenue, and increase sales.
- One of the key pain points for marketers is to know their customers and identify their needs.
- By understanding the customer, marketers can launch a targeted marketing campaign that is tailored for specific needs.
- If data about the customers is available, data science and AI/ML can be applied to perform market segmentation.

B. KMEANS

A. CONCEPT

- K-means is an unsupervised learning algorithm (clustering).
- K-means works by grouping some data points together (clustering) in an unsupervised fashion.
- The algorithm groups observations with similar attribute values together by measuring the Euclidian distance between points.



B. K-MEAN ALGORITHM STEPS

1. Choose number of clusters "K".
2. Select random K points that are going to be the centroids for each cluster.
3. Assign each data point to the nearest centroid, doing so will enable us to create "K" number of clusters.
4. Calculate a new centroid for each cluster.

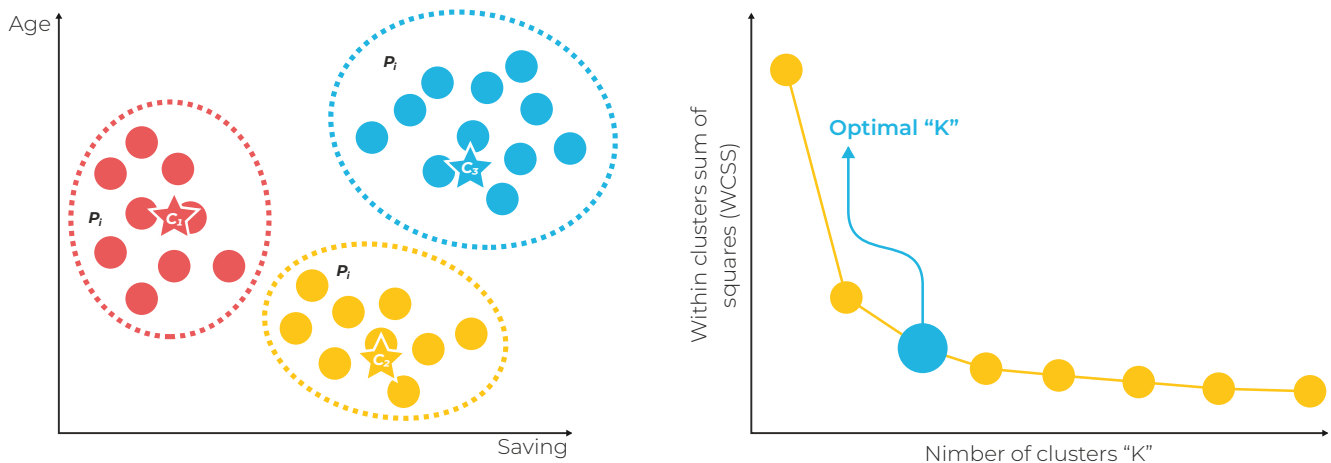
5. Reassign each data point to the new closest centroid.
6. Go to step 4 and repeat.

C. HOW TO CHOOSE OPTIMAL NUMBER OF K?

- Calculate the “Within Cluster Sum of Squares (WCSS)” for various values of K (number of clusters).
- Plot the WCSS vs. K and choose the elbow of the curve as the optimal number of clusters to use.

Within cluster sum of squares (WCSS)

$$= \sum_{P_i \text{ in Cluster 1}} \text{distance}(P_i, C_1)^2 + \sum_{P_i \text{ in Cluster 2}} \text{distance}(P_i, C_2)^2 + \sum_{P_i \text{ in Cluster 3}} \text{distance}(P_i, C_3)^2$$

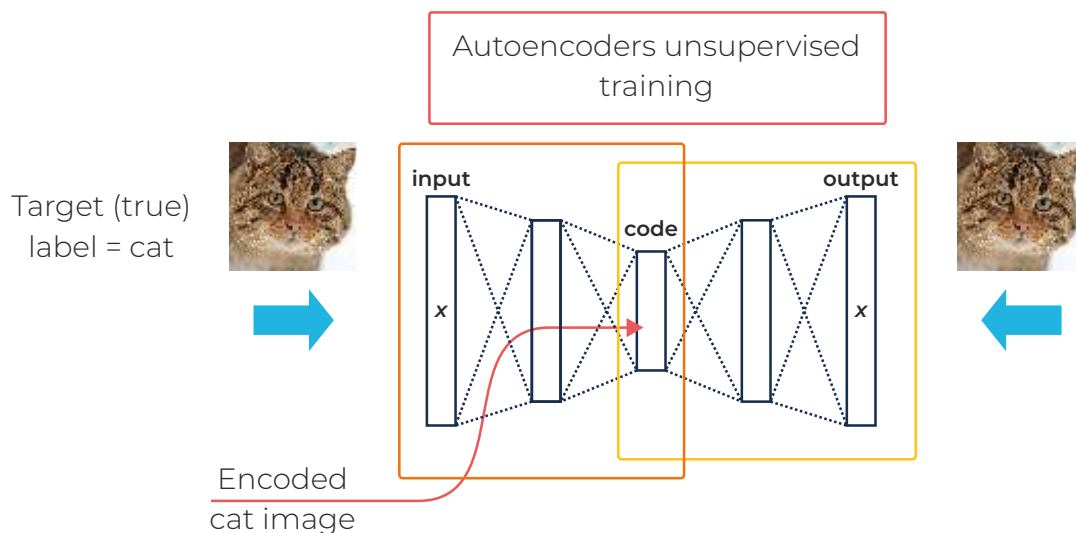


D. HOW TO IMPLEMENT K-MEANS IN SCI-KIT LEARN?

```
>> k = 4 #specify the number of clusters
>> kmeans = KMeans(k)
>> kmeans.fit(X_train)
>> labels = kmeans.labels_
```

C. AUTOENCODERS INTUITION

- Auto encoders are a type of Artificial Neural Networks that are used to perform a task of data encoding (representation learning).
- Auto encoders use the same input data for the input and output, Sounds crazy right!?



- Auto encoders work by adding a bottleneck in the network.
- This bottleneck forces the network to create a compressed (encoded) version of the original input.
- Auto encoders work well if correlations exist between input data (performs poorly if the all input data is independent).

D. AUTOENCODER – INTUITION

```
>>input_df = Input(shape = (37,))
>>x = Dense(50, activation = 'relu')(input_df)
>>x = Dense(500, activation = 'relu', kernel_initializer =
'glorot_uniform')(x)
>>x = Dense(500, activation = 'relu', kernel_initializer =
'glorot_uniform')(x)
```

```
>>x = Dense(2000, activation = 'relu', kernel_initializer =  
'glorot_uniform')(x)  
>>encoded = Dense(8, activation = 'relu', kernel_initializer =  
'glorot_uniform')(x)  
>>x = Dense(2000, activation = 'relu', kernel_initializer =  
'glorot_uniform')(encoded)  
>>x = Dense(500, activation = 'relu', kernel_initializer =  
'glorot_uniform')(x)  
>>decoded = Dense(37, kernel_initializer = 'glorot_uniform')(x)  
  
# autoencoder  
>>autoencoder = Model(input_df, decoded)  
  
# encoder - used for dimensionality reduction  
>>encoder = Model(input_df, encoded)  
  
>>autoencoder.compile(optimizer = 'adam', loss='mean_squared_error')
```

6. CREATIVE AI



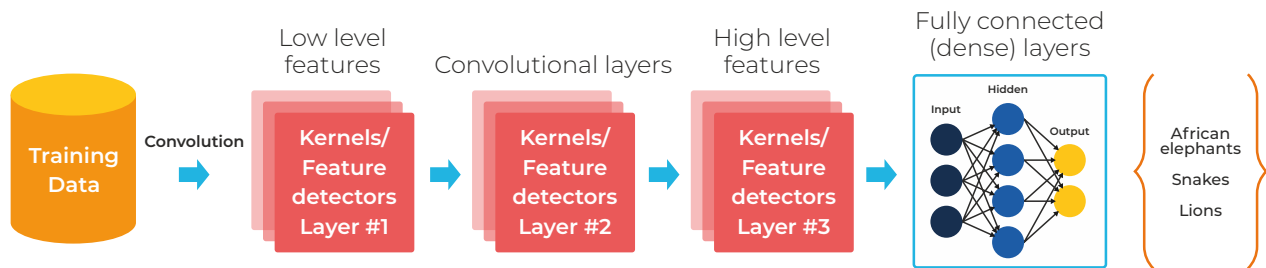
A. CONCEPT

- Creative AI is a new branch of Artificial intelligence in which AI can create paintings, write compelling stories, and compose new music.
- Deep dream is a computer vision algorithm developed by Alex Mordvintsev at Google.
- The algorithm works by creating dream-like effect.
- It's like giving humans an extremely powerful drug!
- As the image is increasingly feed to the network, more weird features will start to pop up.
- Remember when you were a kid looking at the clouds and trying to interpret shapes? This is a horse, here's a dog.
- DeepDream does the same thing by boosting the patterns it sees in a given image based on what it has been trained to see in the past (during training).
- If a network has been trained to see animals in images, it will try to extract animal features in any given image.

B. DEEP DREAM WORKING

- If you feed an image to a CNN, the first layers generally detect lowlevel features such as edges.
- As you go deeper in the network, higher level features are then detected such as faces, trees, and cars.

- “The final few layers assemble those into complete interpretations—these neurons activate in response to very complex things such as entire buildings or trees,” Google’s engineers explain.



C. DEEP DREAM ALGORITHM STEPS

- Forward an image through a trained ANN, CNN, ResNet..etc.
- Select a layer of choice (first layers capture edges, deep layers capture full shapes such as faces).
- Calculate the activations (output) coming out from the layer of interest.
- Calculate the gradient of the activations with respect to the input image.
- Modify the image to increase these activations, and thus enhance the patterns seen by the network resulting in trippy hallucinated image!
- Iterate and repeat over multiple scales.

D. DEEP DREAM ALGORITHM IMPLEMENTATION:

```
>>base_model = tf.keras.applications.InceptionV3(include_top = False,
>>weights = 'imagenet')
>>names = ['mixed3', 'mixed5']
>>layers = [base_model.get_layer(name).output for name in names]
>>deepdream_model = tf.keras.Model(inputs=base_model.input,
outputs=layers) >>img_1 = Image.open("/content/drive/My Drive/Colab
Notebooks/Modern AI Portfolio Builder/Art Creation by AI/sky.jpg")
>>img_2 = Image.open('/content/drive/My Drive/Colab Notebooks/Modern
AI Portfolio Builder/Art Creation by AI/sea.jpg')
>>image = Image.blend(img_1, img_2, 0.5)
>>Sample_Image1 = tf.keras.preprocessing.image.load_img('img_0.jpg')
>>Sample_Image1 =
tf.keras.preprocessing.image.img_to_array(Sample_Image1)
>>Sample_Image1.shape
>>Sample_Image1 = tf.expand_dims(Sample_Image1, axis = 0)
>>activations = deepdream_model(Sample_Image1)

>>def deepdream(model, image, step_size):
>>    with tf.GradientTape() as tape:
>>        tape.watch(image)
>>        loss = calc_loss(image, model)
>>        gradients = tape.gradient(loss, image)
>>        gradients /= tf.math.reduce_std(gradients)
>>        image = image + gradients * step_size
>>        image = tf.clip_by_value(image, -1, 1)
>>    return loss, image

>>def run_deep_dream_simple(model, image, steps=100, tep_size=0.01):
>>    image =
tf.keras.applications.inception_v3.preprocess_input(image)
>>    for step in range(steps):
>>        loss, image = deepdream(model, image, step_size)
>>        if step % 100 == 0:
>>            plt.figure(figsize=(12,12))
>>            plt.imshow(deprocess(image))
>>            plt.show()
>>            print ("Step {}, loss {}".format(step, loss))
>>    plt.figure(figsize=(12,12))
>>    plt.imshow(deprocess(image))
>>    plt.show()
>>    return deprocess(image)

>>def deprocess(image):
>>    image = 255*(image + 1.0)/2.0
>>    return tf.cast(image, tf.uint8)

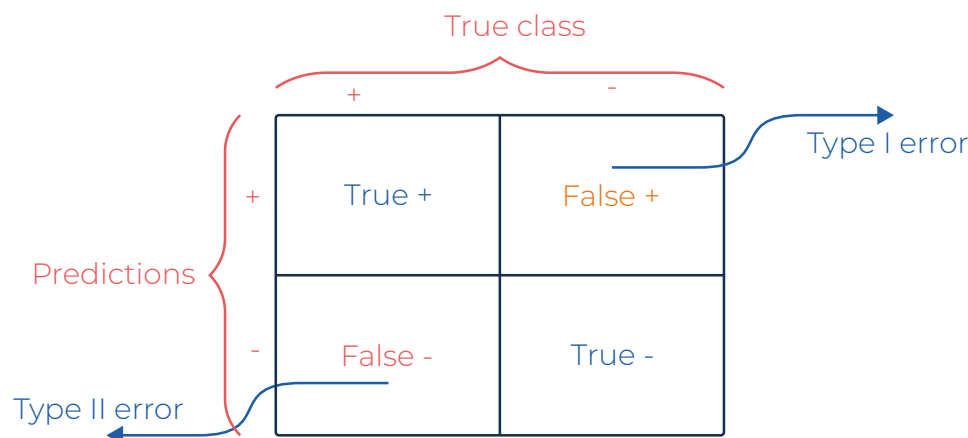
>>Sample_Image =
tf.keras.preprocessing.image.img_to_array(Sample_Image) >>dream_img =
run_deep_dream_simple(model=deepdream_model, image=Sample_Image1,
steps=2000, step_size=0.001)
```


7. CONFUSION MATRIX

A. CONCEPT

A confusion matrix is used to describe the performance of a classification model:

- True positives (TP): cases when classifier predicted TRUE (has a disease), and correct class was TRUE (patient has disease).
- True negatives (TN): cases when model predicted FALSE (no disease), and correct class was FALSE (patient does not have disease).
- False positives (FP) (Type I error): classifier predicted TRUE, but correct class was FALSE (patient does not have disease).
- False negatives (FN) (Type II error): classifier predicted FALSE (patient do not have disease), but they actually do have the disease.



- Classification Accuracy = $(TP+TN) / (TP + TN + FP + FN)$.
- Misclassification rate (Error Rate) = $(FP + FN) / (TP + TN + FP + FN)$.
- Precision = $TP / \text{Total TRUE Predictions} = TP / (TP+FP)$ (When model predicted TRUE class, how often did it get it right?).

- $\text{Recall} = \text{TP} / \text{Actual TRUE} = \text{TP} / (\text{TP} + \text{FN})$ (when the class was actually TRUE, how often did the classifier get it right?).

B. CONFUSION MATRIX IN SKLEARN

```
>> from sklearn.metrics import classification_report,
confusion_matrix
>> y_predict_test = classifier.predict(X_test)
>> cm = confusion_matrix(y_test, y_predict_test)
>> sns.heatmap(cm, annot=True)
```

C. CLASSIFICATION REPORT

```
>> from sklearn.metrics import classification_report
>> print(classification_report(y_test, y_pred))
```

8. REGRESSION MACHINE LEARNING MODELS METRICS

A. MEAN ABSOLUTE ERROR (MAE)

- Mean Absolute Error (MAE) is obtained by calculating the absolute difference between the model predictions and the true (actual) values.
- MAE is a measure of the average magnitude of error generated by the regression model.
- The mean absolute error (MAE) is calculated as follows:

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

- MAE is calculated by following these steps:
 1. Calculate the residual for every data point
 2. Calculate the absolute value (to get rid of the sign)
 3. Calculate the average of all residuals
- If MAE is zero, this indicates that the model predictions are perfect.

B. MEAN SQUARE ERROR (MSE)

- Mean Square Error (MSE) is very similar to the Mean Absolute Error (MAE) but instead of using absolute values, squares of the difference between the model predictions and the training dataset (true values) is being calculated.
- MSE values are generally large compared to the MAE since the residuals are being squared.
- In case of data outliers, MSE will become much larger compared to MAE.
- In MSE, error increases in a quadratic fashion while the error increases in proportional fashion in MAE.
- The MSE is calculated as follows:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- MAE is calculated by following these steps:
 1. Calculate the residual for every data point
 2. Calculate the squared value of the residuals
 3. Calculate the average of all residuals



C. ROOT MEAN SQUARE ERROR (RMSE)

- Root Mean Square Error (RMSE) represents the standard deviation of the residuals (i.e.: differences between the model predictions and the true values (training data)).
- RMSE can be easily interpreted compared to MSE because RMSE units match the units of the output.
- RMSE provides an estimate of how large the residuals are being dispersed.
- The MSE is calculated as follows:

$$MSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

- RMSE is calculated by following these steps:
 1. Calculate the residual for every data point
 2. Calculate the squared value of the residuals
 3. Calculate the average of the squared residuals
 4. Obtain the square root of the result



D. MEAN ABSOLUTE PERCENTAGE ERROR (MAPE)

- MAE values can range from 0 to infinity which makes it difficult to interpret the result as compared to the training data.
- Mean Absolute Percentage Error (MAPE) is the equivalent to MAE but provides the error in a percentage form and therefore overcomes MAE limitations.
- MAPE might exhibit some limitations if the data point value is zero (since there is division operation involved).
- The MAPE is calculated as follows:

$$MAPE = \frac{100\%}{n} \sum_{i=1}^n |(y_i - \hat{y}_i)/y_i|$$

E. MEAN PERCENTAGE ERROR (MPE)

- MPE is similar to MAPE but without the absolute operation.
- MPE is useful to provide an insight of how many positive errors as compared to negative ones.
- The MPE is calculated as follows:

$$MPE = \frac{100\%}{n} \sum_{i=1}^n (y_i - \hat{y}_i)/y_i$$