INTRODUCTION TO EMOTION AI

- **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/**

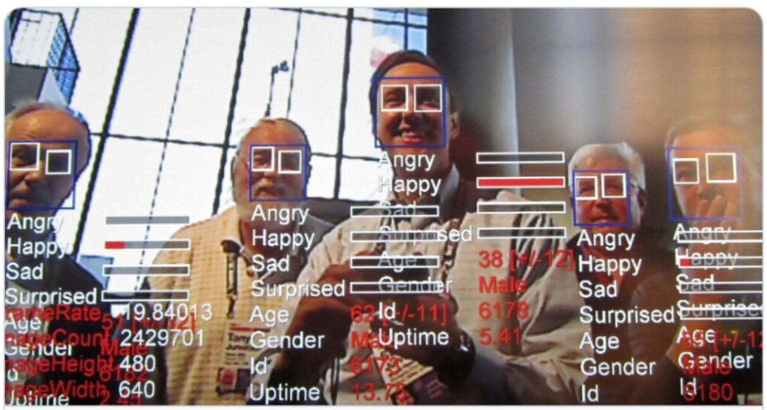
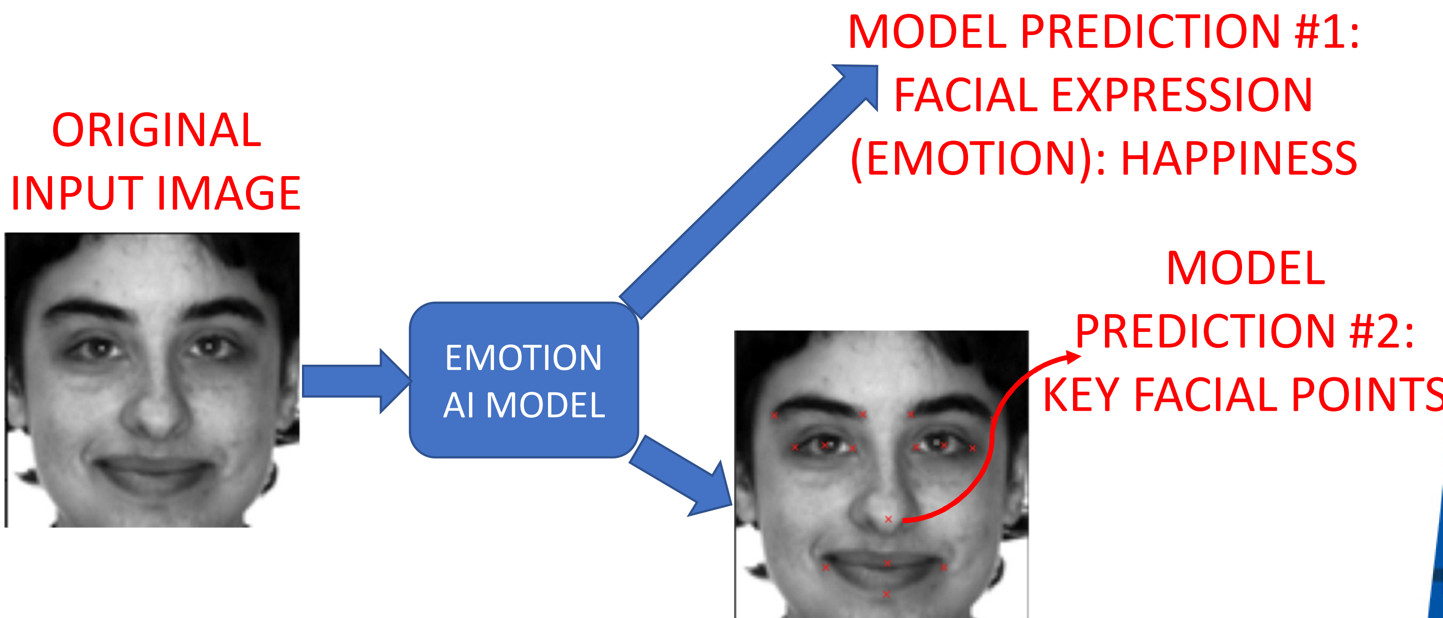


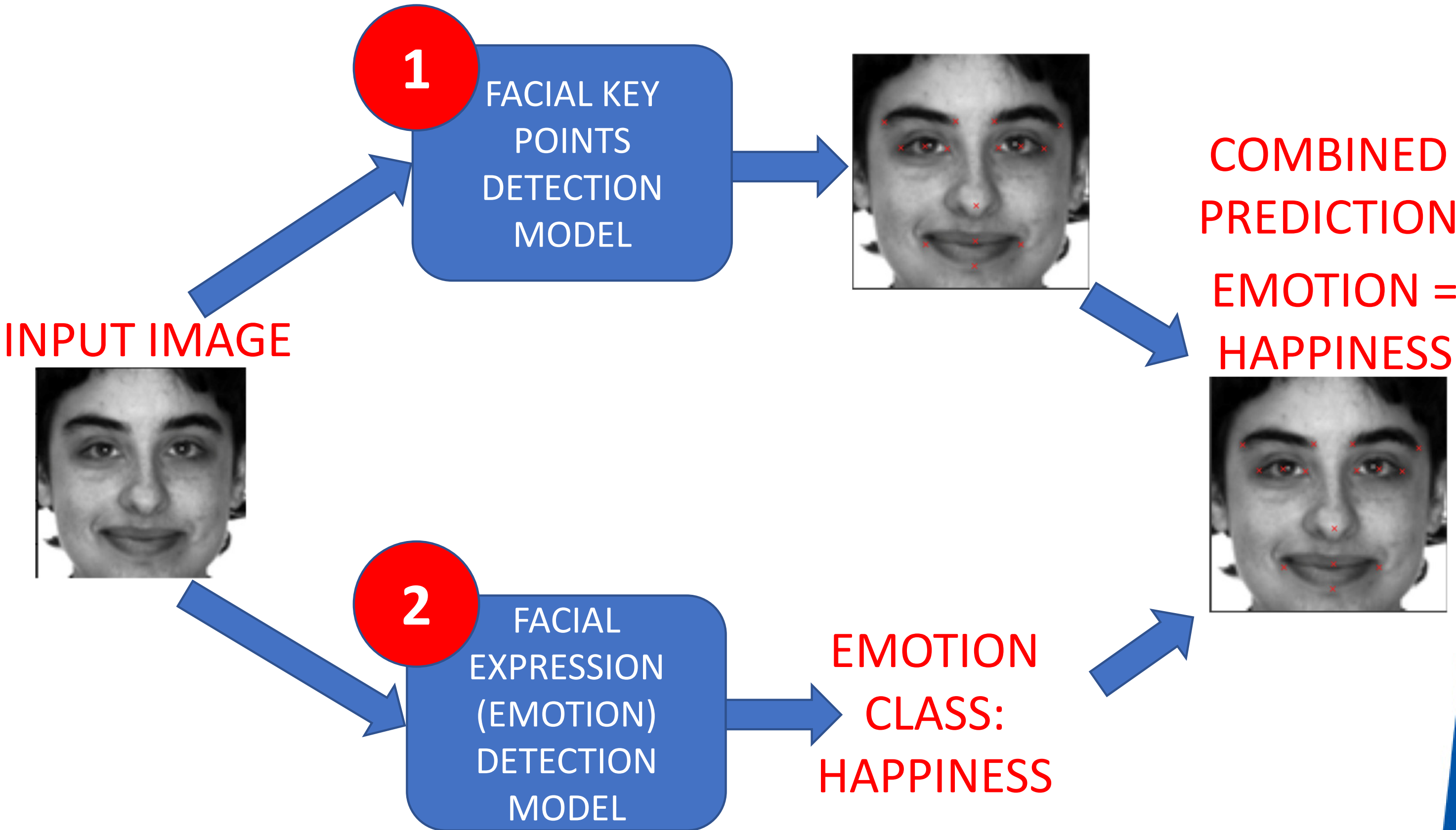
Photo Credit: https://en.wikipedia.org/wiki/File:11vRwNSw.jpg
Photo Credit: https://www.flickr.com/photos/jurvetson/49352718206

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.
- The team has collected more than 20000 facial images, with their associated facial expression labels and around 2000 images with their facial key-point annotations.



PROJECT OVERVIEW: PIPELINE TO DETECT KEYPOINTS & EMOTIONS



PART 1. KEY FACIAL POINTS DETECTION

- In part #1, we will create a deep learning model based on Convolutional Neural Network and Residual Blocks to predict facial key-points.

Data Source: https://www.kaggle.com/c/facial-keypoints-detection/data

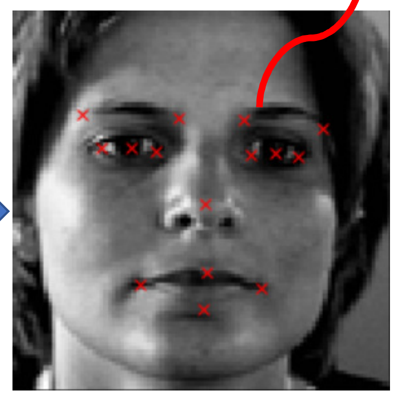
PART 1. KEY FACIAL POINTS DETECTION

- The dataset consists of x and y coordinates of 15 facial key points.
- Input Images are 96 x 96 pixels.
- Images consist of only one color channel (gray-scale images).

INPUT IMAGE

TRAINED KEY FACIAL POINTS DETECTOR MODEL

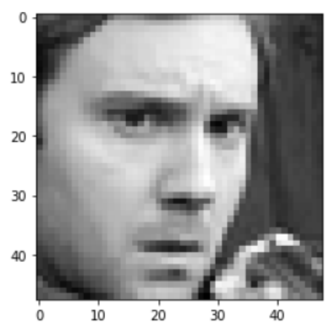
KEY FACIAL POINTS COORDINATES



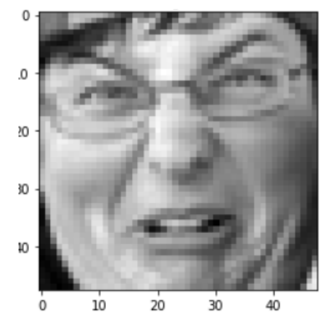
PART 2. FACIAL EXPRESSION (EMOTION) DETECTION

- The second model will classify people’s emotion.
- Data contains images that belong to 5 categories:
    - 0 = Angry
    - 1 = Disgust
    - 2 = Sad
    - 3 = Happy
    - 4 = Surprise

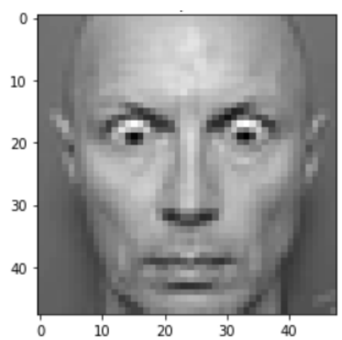
ANGER



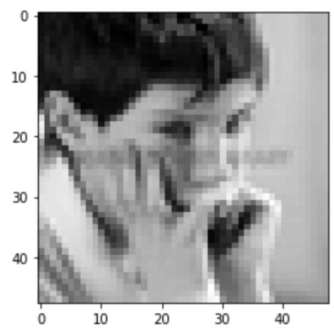
DIGUST



SURPRISE!



SAD



HAPPINESS

Data is source from Kaggle: https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data

PART 2. FACIAL EXPRESSION (EMOTION) DETECTION

INPUT IMAGE

48

48

CLASSIFIER (DEEP LEARNING MODEL)

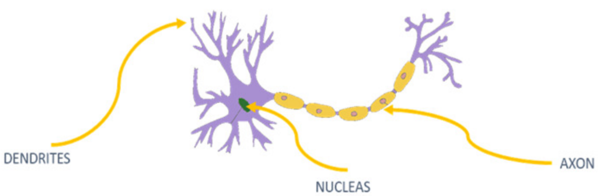
TARGET CLASSES

Angry
Disgust
Sad
Happiness
Surprise

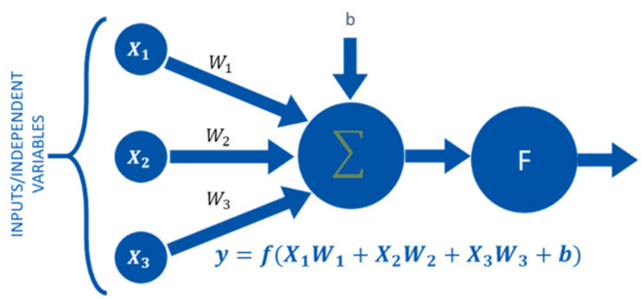
NEURON MATHEMATICAL MODEL

- The brain has over 100 billion neurons communicating through electrical & chemical signals. Neurons communicate with each other and help us see, think, and generate ideas.
- Human brain learns by creating connections among these neurons. ANNs are information processing models inspired by the human brain.
- The neuron collects signals from input channels named dendrites, processes information in its nucleus, and then generates an output in a long thin branch called axon.

HUMAN NEURON



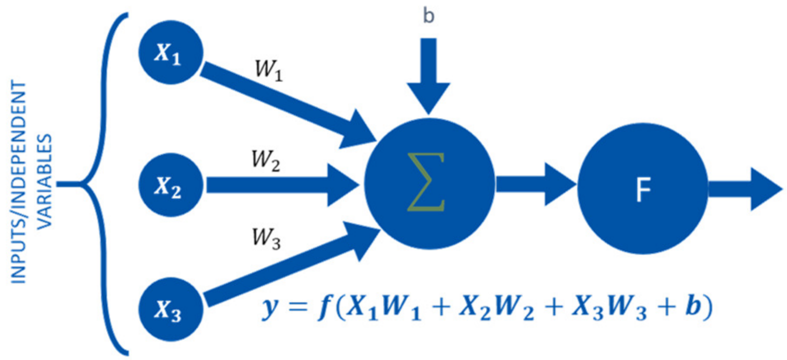
ARTIFICIAL NEURON

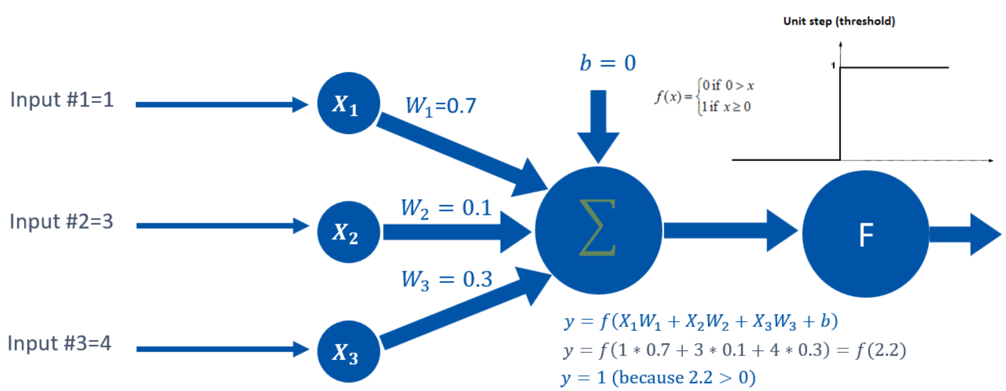


- Photo Credit: https://en.wikipedia.org/wiki/File:Neuron-no_labels2.png
- Photo Credit: https://www.flickr.com/photos/alansimpsonme/34752491090

NEURON MATHEMATICAL MODEL: EXAMPLE

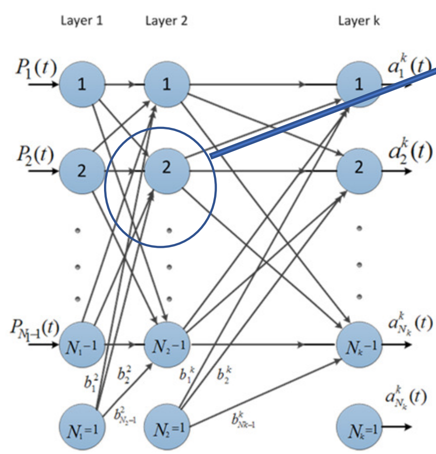
- Bias allows to shift the activation function curve up or down.
- Number of adjustable parameters = 4 (3 weights and 1 bias).
- Activation function "F".

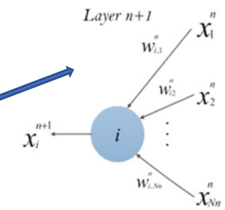




MULTI-LAYER PERCEPTRON NETWORK

- Let’s connect multiple of these neurons in a multi-layer fashion.
- The more hidden layers, the more “deep” the network will get.





$$x_i^{n+1}(t) = \varphi\left(\sum_{j=1}^{N_n} w_{i,j}^n \, x_j^n(t)\right)$$

Node $(n+1,\ i)$ representation

$$P = \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_{N_1} \end{bmatrix}$$

Non-Linear Sigmoid Activation function

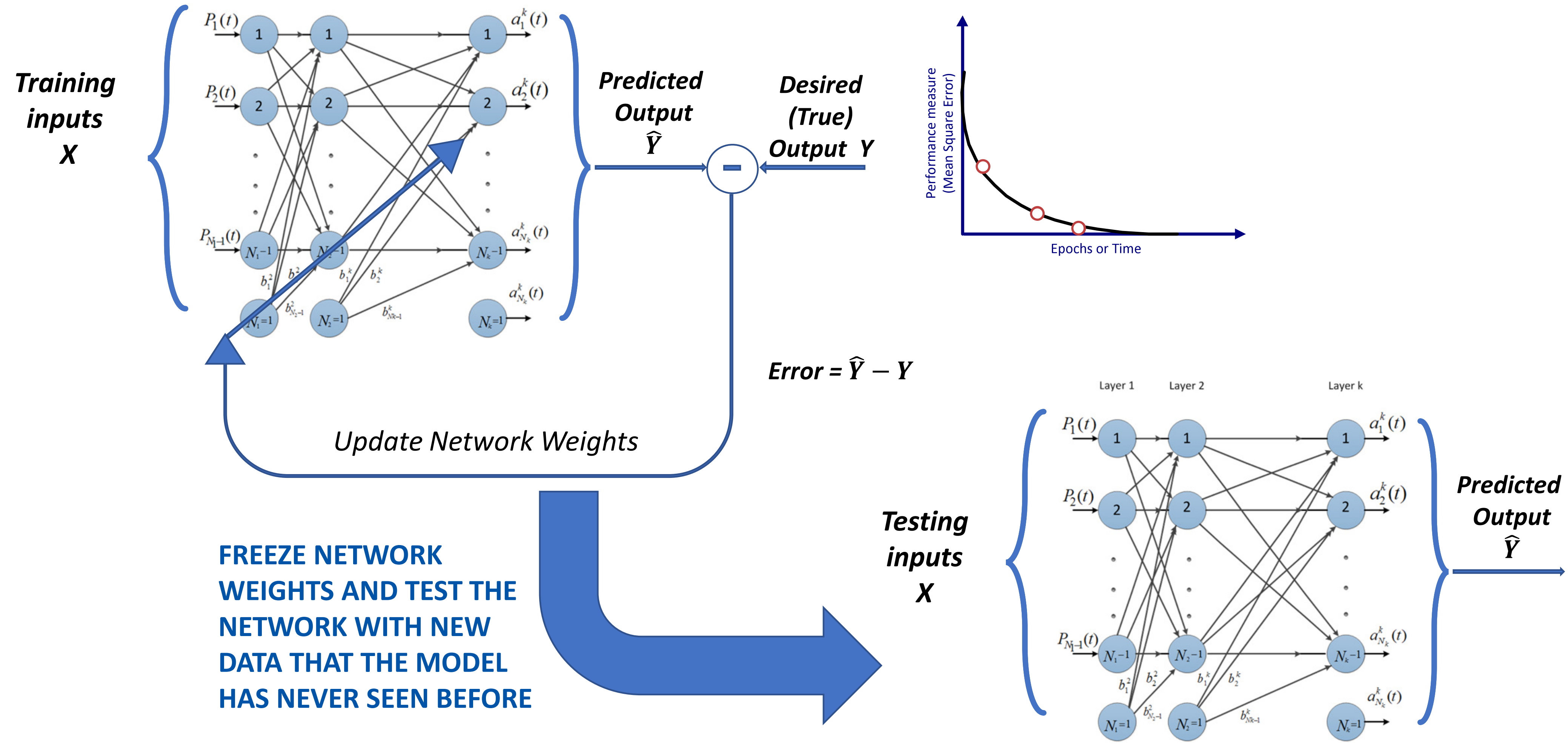
$$\varphi(w) = \frac{1}{1 + e^{-w}}$$

m: number of neurons in the hidden layer

N_1*: number of inputs*

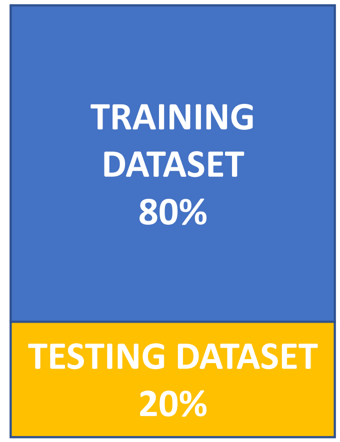
$$\begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1,N_1} \\ W_{21} & W_{22} & & W_{2,N_1} \\ \vdots & & \ddots & \vdots \\ W_{m-1,1} & W_{m-1,2} & \cdots & W_{m-1,N_1} \\ W_{m,1} & W_{m,2} & & W_{m,N_1} \end{bmatrix}$$

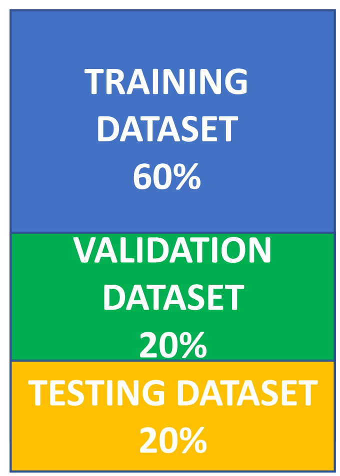
ANNs TRAINING & TESTING PROCESSES



DIVIDE DATA INTO TRAINING AND TESTING

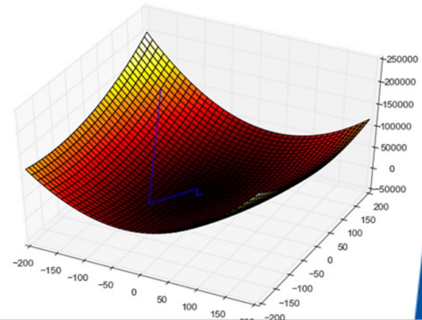
- Data set is generally divided into 80% for training and 20% for testing.
- Sometimes, we might include cross validation dataset as well and then we divide it into 60%, 20%, 20% segments for training, validation, and testing, respectively (numbers may vary).
    1. Training set: used for gradient calculation and weight update.
    2. Validation set:
        - used for cross-validation to assess training quality as training proceeds.
        - Cross-validation is implemented to overcome over-fitting which occurs when algorithm focuses on training set details at cost of losing generalization ability.
    3. Testing set: used for testing trained network.

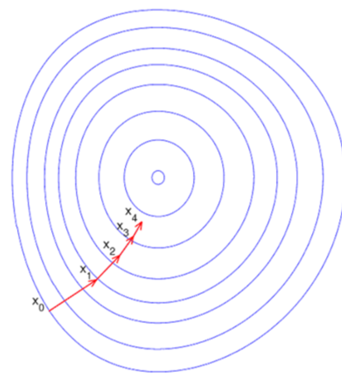




GRADIENT DESCENT

- Gradient descent is an optimization algorithm used to obtain the optimized network weight and bias values
- It works by iteratively trying to minimize the cost function
- It works by calculating the gradient of the cost function and moving in the negative direction until the local/global minimum is achieved
- If the positive of the gradient is taken, local/global maximum is achieved
- The size of the steps taken are called the learning rate
- If learning rate increases, the area covered in the search space will increase so we might reach global minimum faster
- However, we can overshoot the target
- For small learning rates, training will take much longer to reach optimized weight values





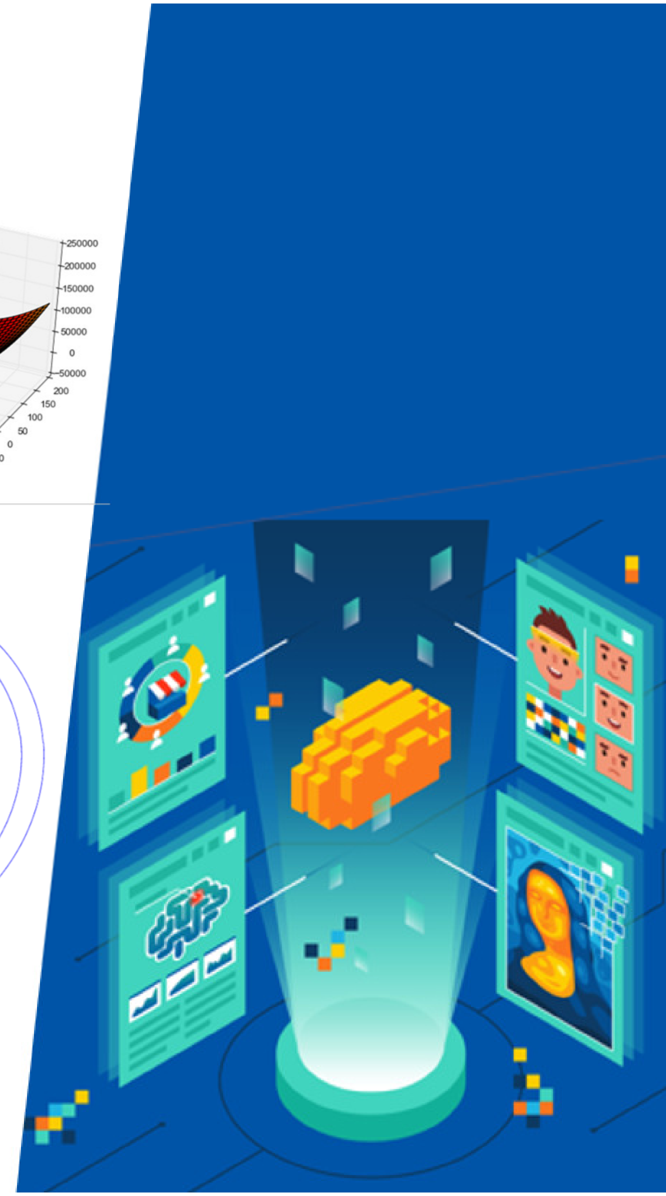


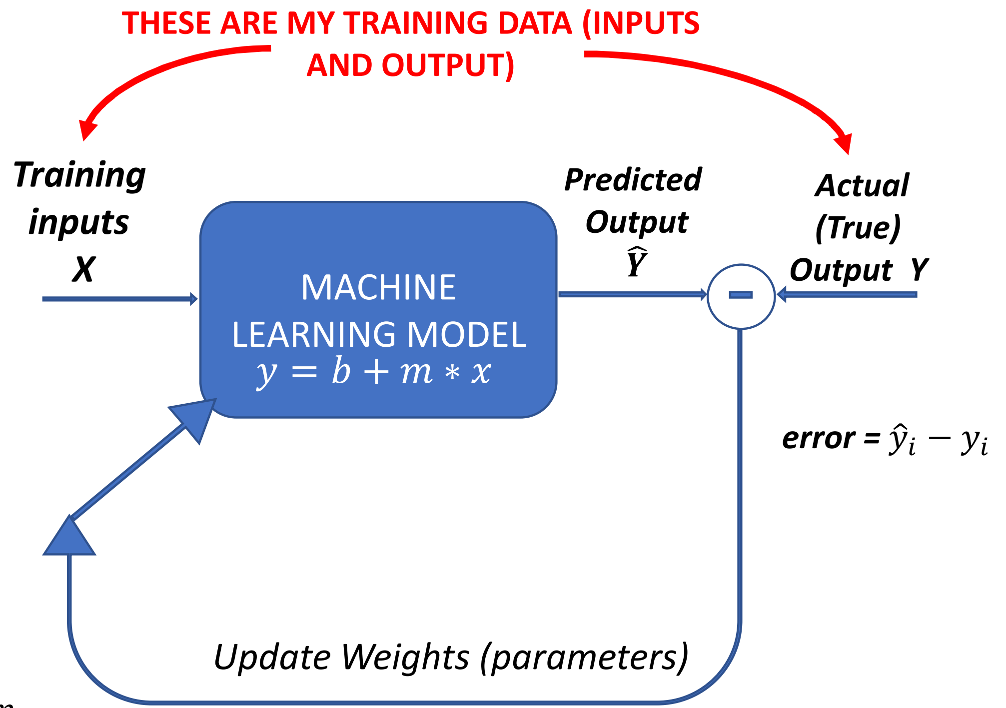
Photo Credit: https://commons.wikimedia.org/wiki/File:Gradient_descent_method.png
Photo Credit: https://commons.wikimedia.org/wiki/File:Gradient_descent.png

GRADIENT DESCENT

- Let's assume that we want to obtain the optimal values for parameters 'm' and 'b'.

$$y = \boxed{b} + \boxed{m} * x$$

GOAL IS TO FIND
BEST PARAMETERS

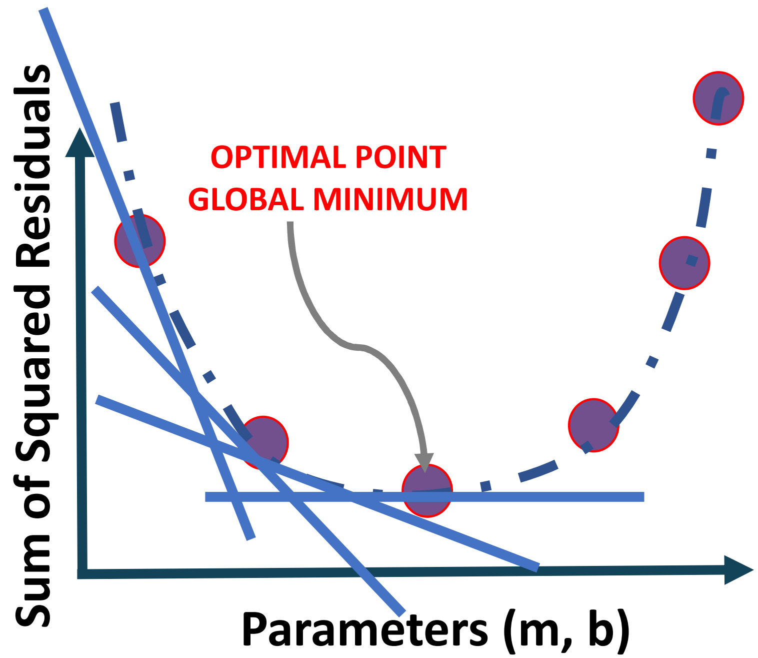


- We need to first formulate a loss function as follows:

$$Cost\ Function\ f(m,b) = \frac{1}{N}\sum_{i=1}^{n}(error)^2 = \frac{1}{N}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

GRADIENT DESCENT

$$Loss\ Function\ f(m,b) = \frac{1}{N}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$



GRADIENT DESCENT WORKS AS FOLLOWS:

1. Calculate the gradient (derivative) of the Loss function $\frac{\partial loss}{\partial w}$
2. Pick random values for weights (m, b) and substitute
3. Calculate the step size (how much are we going to update the parameters?)

$$Step\ size\ = learning\ rate * gradient = \alpha * \frac{\partial loss}{\partial w}$$

4. Update the parameters and repeat

$$new\ weight\ = old\ weight - step\ size$$

$$w_{new} = w_{old} - \alpha * \frac{\partial loss}{\partial w}$$

*Note: in reality, this graph is 3D and has three axes, one for m, b and sum of squared residuals

CONVOLUTIONAL NEURAL NETWORKS: ENTIRE NETWORK OVERVIEW

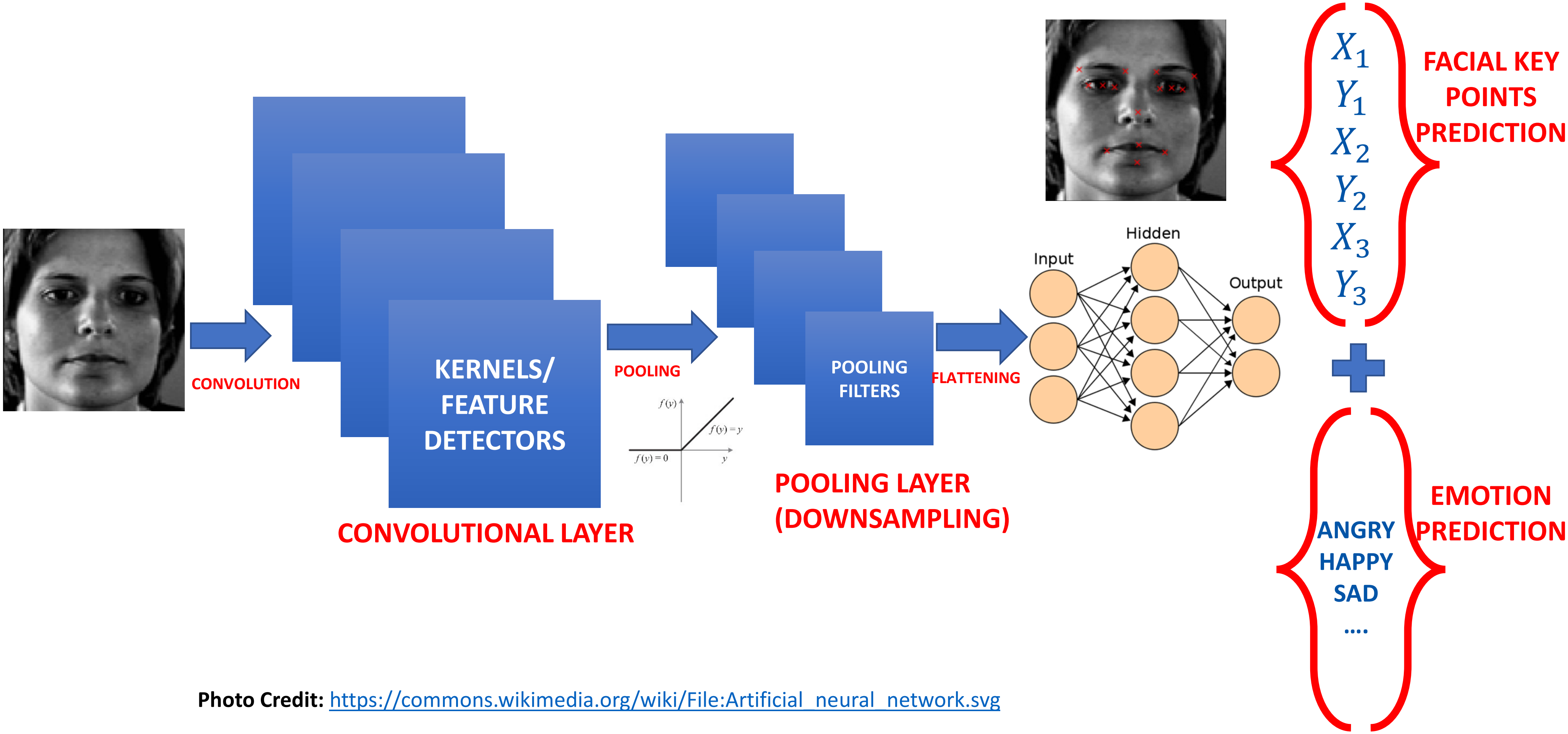
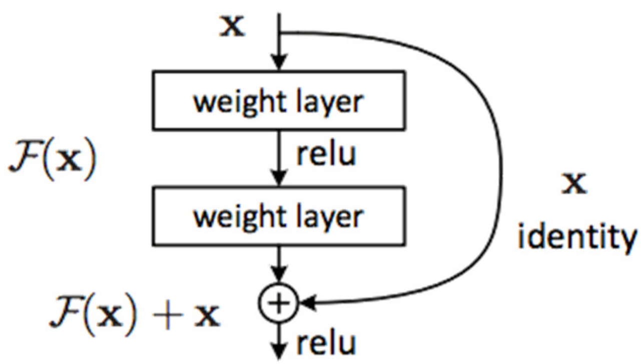


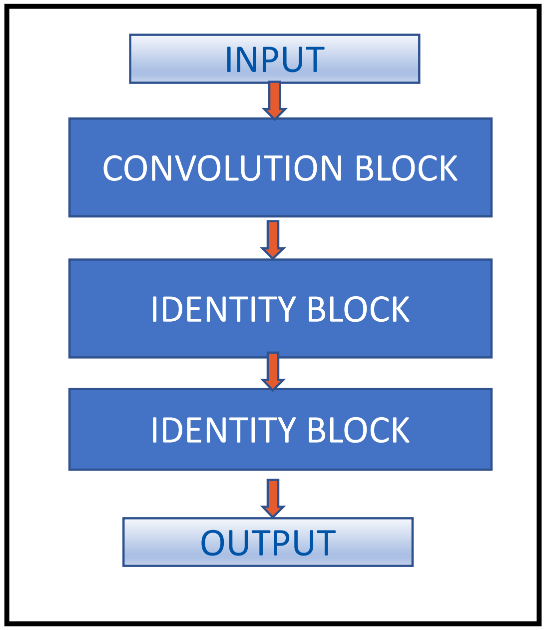
Photo Credit: https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg

RESNET (RESIDUAL NETWORK)

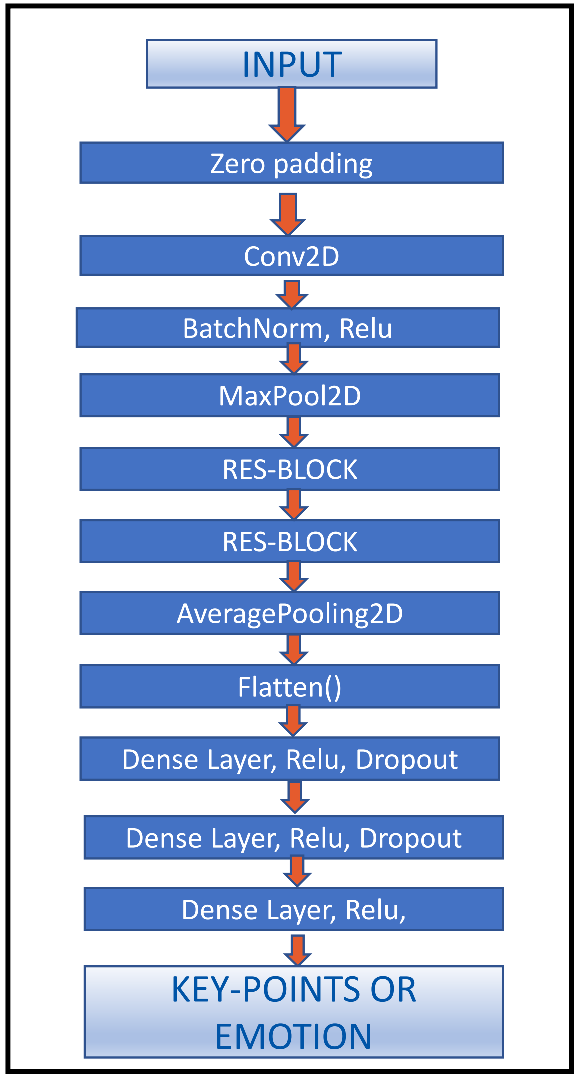
- As CNNs grow deeper, vanishing gradient tend to occur which negatively impact network performance.
- Vanishing gradient problem occurs when the gradient is back-propagated to earlier layers which results in a very small gradient.
- Residual Neural Network includes “skip connection” feature which enables 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.



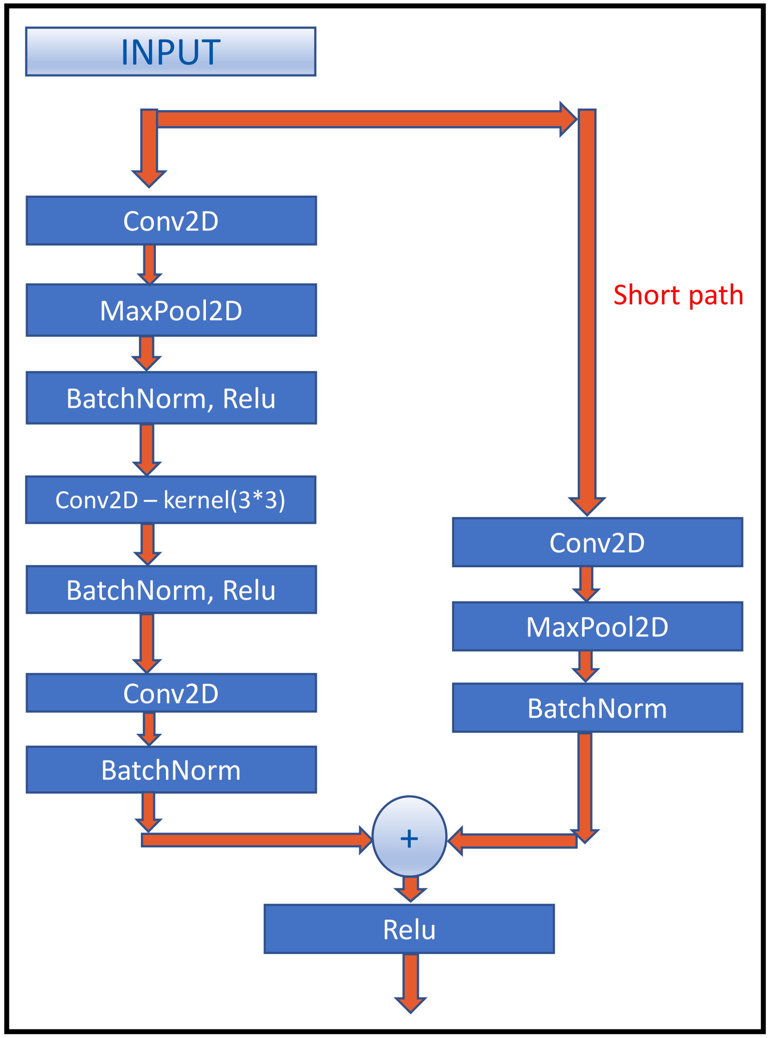
RES-BLOCK



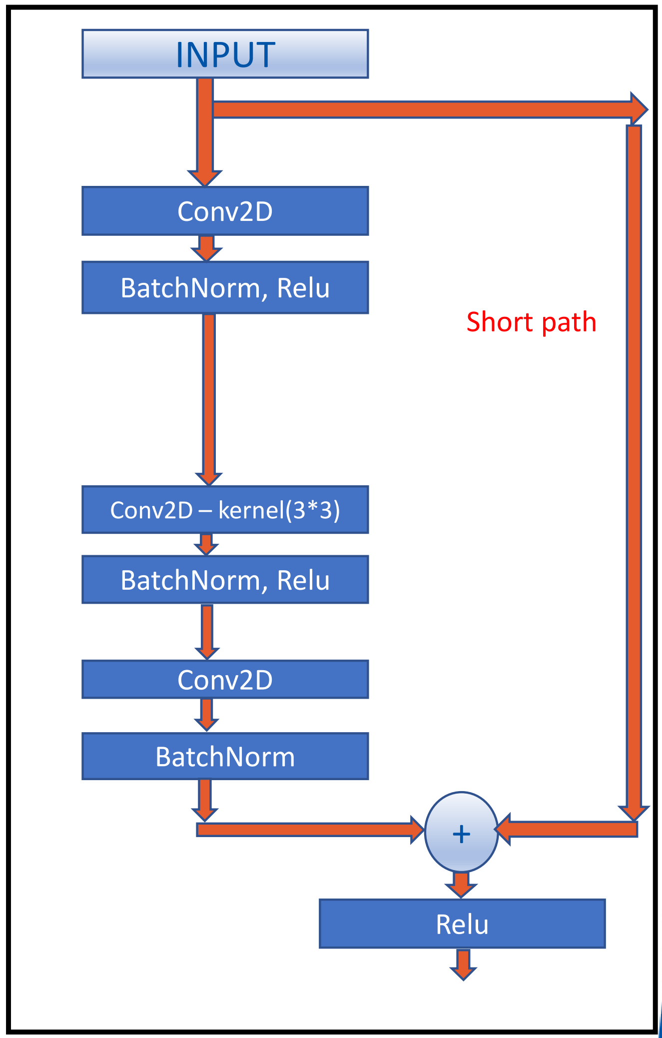
FINAL MODEL



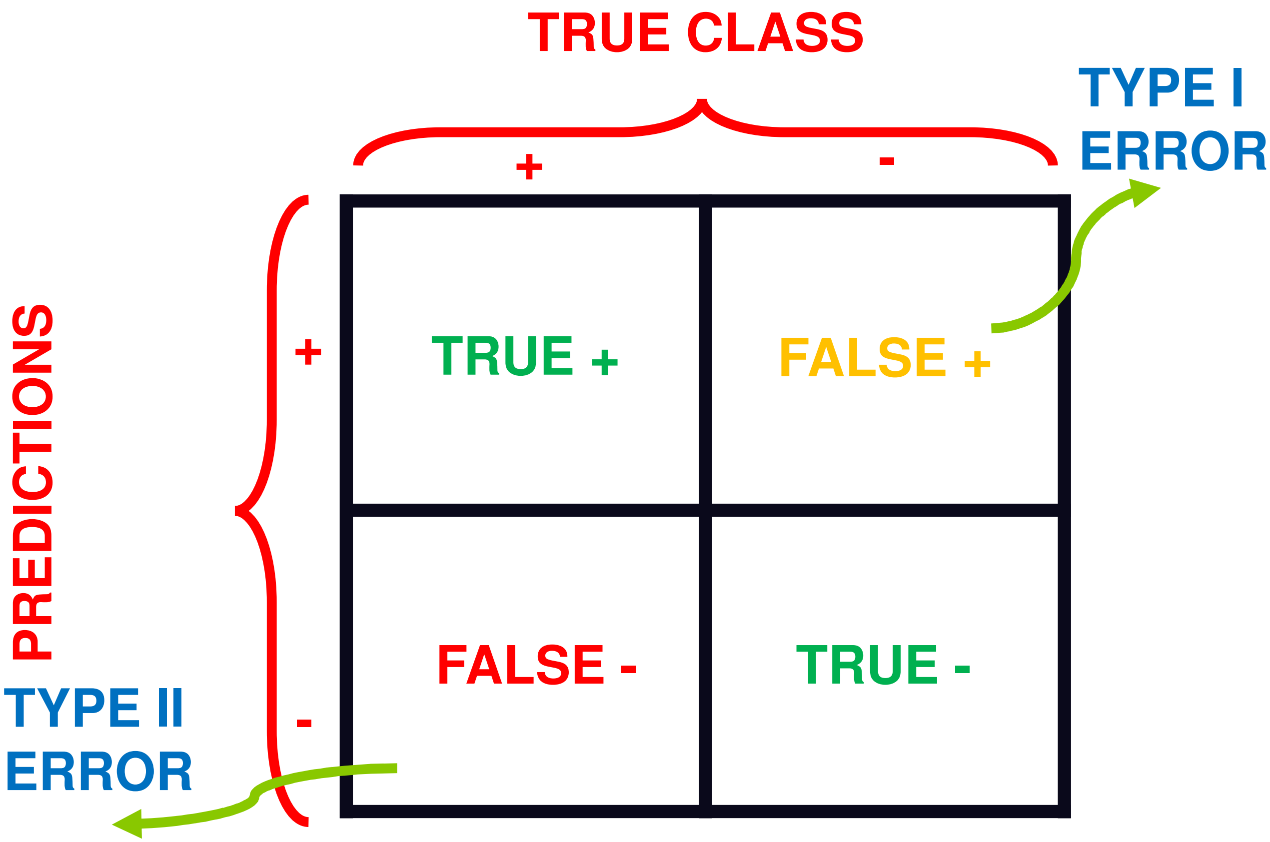
CONVOLUTION BLOCK



IDENTITY BLOCK



CONFUSION MATRIX



DEFINITIONS AND KPIS

- A confusion matrix is used to describe the performance of a classification model:
  - **True positives (TP): cases when classifier predicted TRUE (they have the 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 do not have disease).**
  - **False positives (FP) (Type I error): classifier predicted TRUE, but correct class was FALSE (patient did 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/Total TRUE Predictions = TP/ (TP+FP) (When model predicted TRUE class, how often was it right?)**
  - **Recall = TP/ Actual TRUE = TP/ (TP+FN) (when the class was actually TRUE, how often did the classifier get it right?)**

PRECISION Vs. RECALL EXAMPLE

TRUE CLASS

PREDICTIONS	+	-
+	TP = 1	FP = 1
-	FN = 8	TN = 90

FACTS:
100 PATIENTS TOTAL
91 PATIENTS ARE HEALTHY
9 PATIENTS HAVE CANCER

- Accuracy is generally misleading and is not enough to assess the performance of a classifier.
- Recall is an important KPI in situations where:
  - Dataset is highly imbalanced; cases when you have small cancer patients compared to healthy ones.

- Classification Accuracy = (TP+TN) / (TP + TN + FP + FN) = 91%
- Precision = TP/Total TRUE Predictions = TP/ (TP+FP) = ½=50%
- Recall = TP/ Actual TRUE = TP/ (TP+FN) = 1/9 = 11%

MODEL DEPLOYMENT USING TENSORFLOW SERVING:

- Let's assume that we already trained our model and it is generating good results on the testing data.
- Now, we want to integrate our trained Tensorflow model into a web app and deploy the model in production level environment.
- The following objective can be obtained using 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".

RUNNING 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.**

MAKING REQUEST IN TENSORFLOW SERVING:

- In-order to make prediction using TensorFlow Serving, we need to pass the inference requests (image data) as a JSON object.
- 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).
- Finally, we get the prediction from the post request made to the deployed model and then use argmax function to find the predicted class.