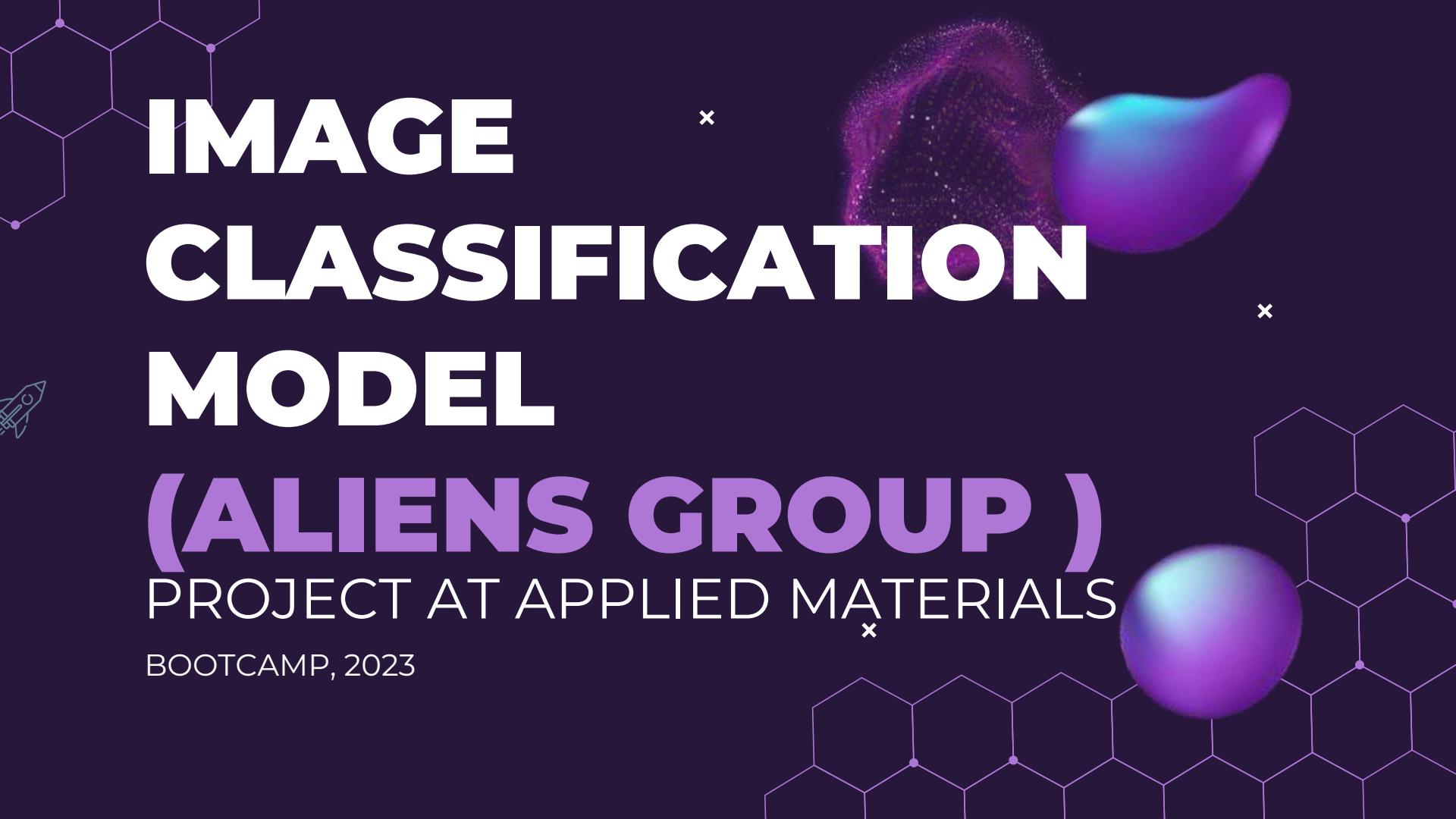


IMAGE CLASSIFICATION MODEL **(ALIENS GROUP)**



The background features a dark purple gradient with abstract white hexagonal patterns resembling a molecular or crystal lattice. A small white rocket ship icon is located on the left side. In the upper right, there is a large, semi-transparent sphere composed of numerous small dots, transitioning from blue at the top to red at the bottom. Below it, two larger, solid-colored spheres (one blue, one red) are partially visible. The overall aesthetic is futuristic and scientific.

PROJECT AT APPLIED MATERIALS

BOOTCAMP, 2023

OUR GROUP

We really enjoyed collaborating on this project specifically and in the bootcamp in general. We felt that each girl brought their own strengths and talents to contribute to the success of the program and it was a very positive learning experience for each of us.

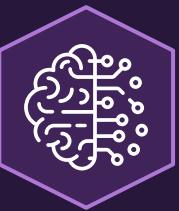


OUR PLAYERS:

x



Kaila Hartman



Ester Yakov



Tamar Gurwicz

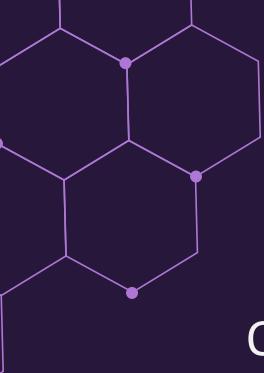


Rachel Braverman

+

x





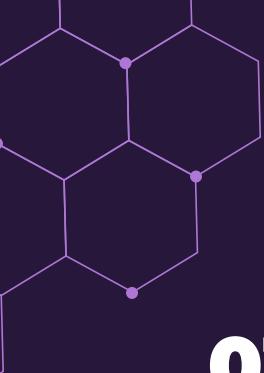
x



OVERVIEW:

Our image classification application employs advanced machine learning techniques to accurately categorize images into 13 distinct classes. Built upon the robust CIFAR-10 dataset and some of the CIFAR-100 dataset, which comprises a diverse range of images, our app provides intelligent classification capabilities that cater to various visual domains.

By integrating a range of powerful tools, including deep learning and transfer learning, we empower our app to make accurate predictions



x

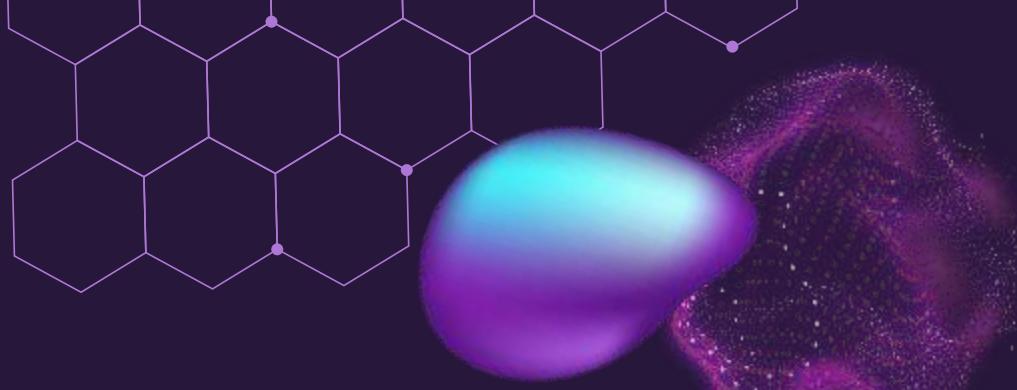
STAGES:



- 01 PREPARE DATA**
- 02 TRAIN MODEL**
- 03 VISUALIZE**
- 04 DEPLOYING & MONITORING**

01

PREPARING THE DATA



x



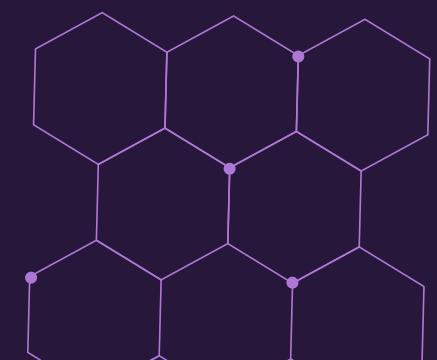
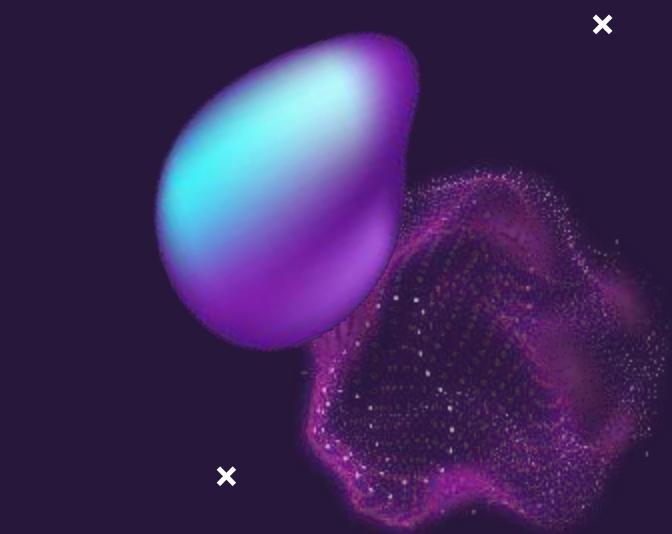
+





“You can have data
without information, but
you cannot have*
information without
data.”

— Daniel Keys Moran.



×

STEPS INCLUDED:



DOWNLOAD

Data from different sources

AUGMENTATION

In order to have balanced data.



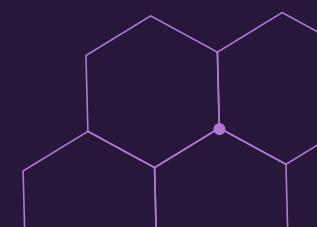
RESIZING+ normalization

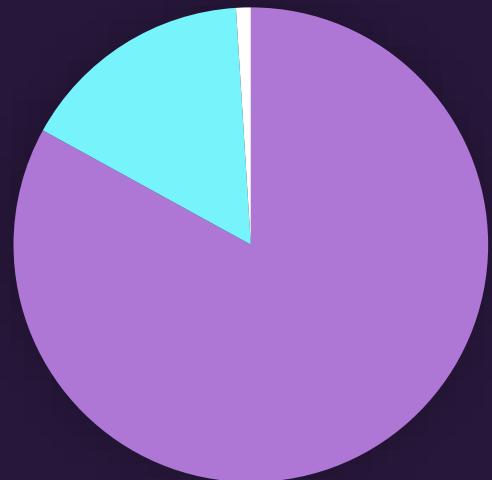
Each additional picture has to have the same size as cifar images.
And needs to be normalized



IMG -> NP ARRAY!

Our model will be trained on an NP array.



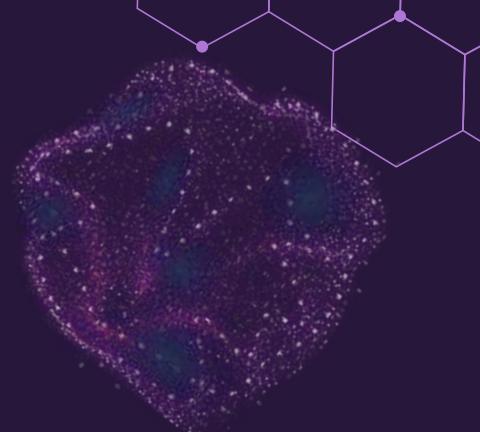


Data Source:

83% CIFAR-10

16% CIFAR-100

**1% Additional
pictures**



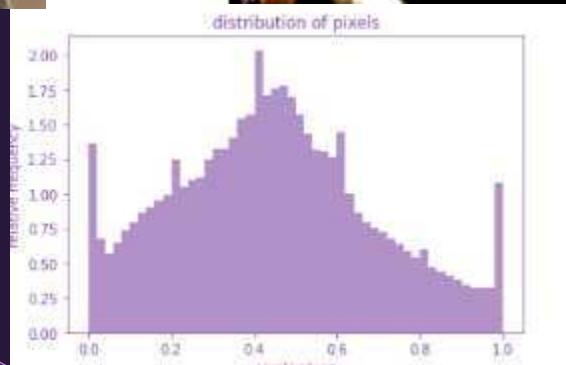
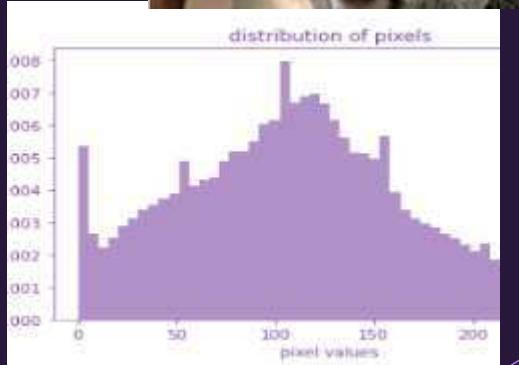
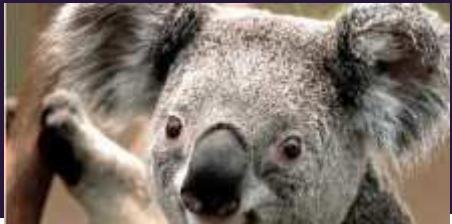
x

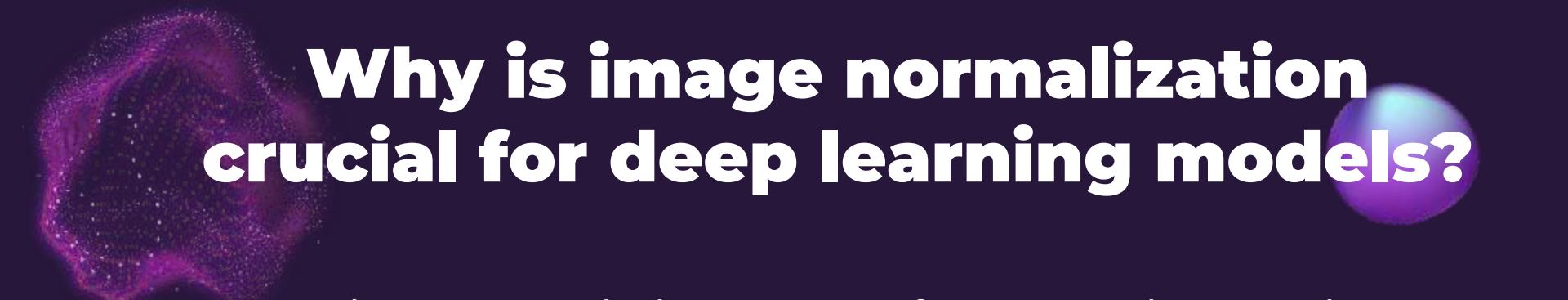
Normalization:

x

What is image normalization?

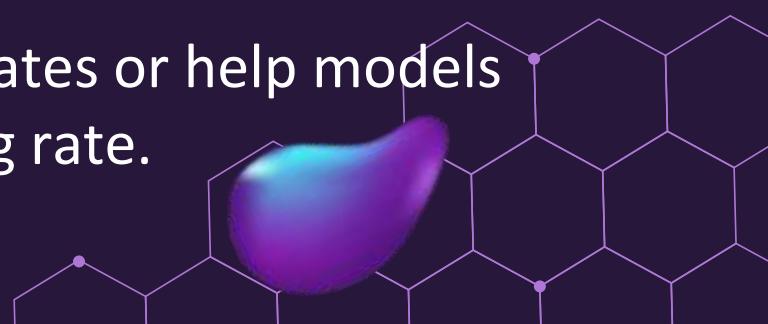
Image normalization scales pixel values from [0, 255] to [0, 1] by dividing each pixel by 255.





Why is image normalization crucial for deep learning models?

-  Normalization can help training of our neural networks as the different features are on a similar scale
-  which helps to stabilize the gradient descent step
-  allowing us to use larger learning rates or help models converge faster for a given learning rate.



Resampling:



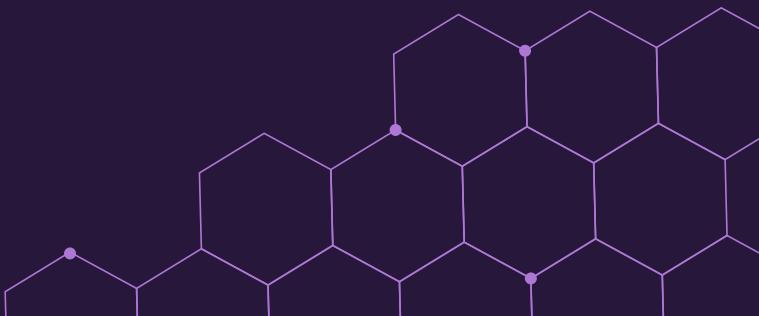
In order to train a neural network effectively we need to have all the images the same size

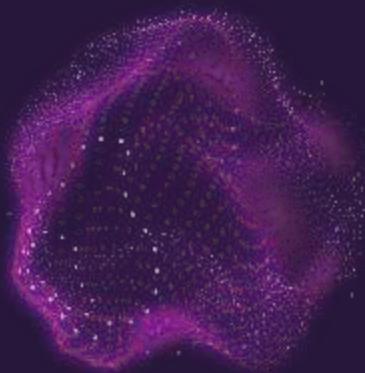


Because the cifar10 images are 32X32 we need to resample the images which are added to the training process.

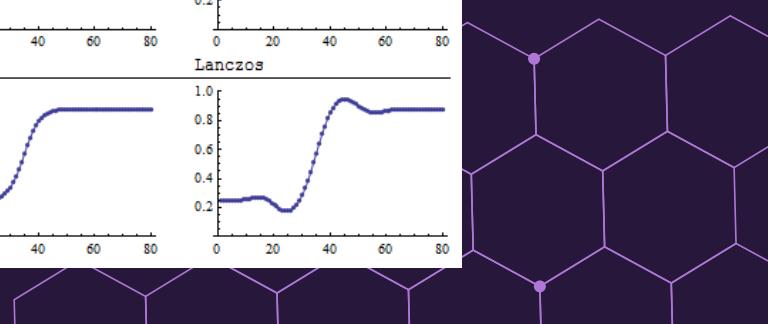
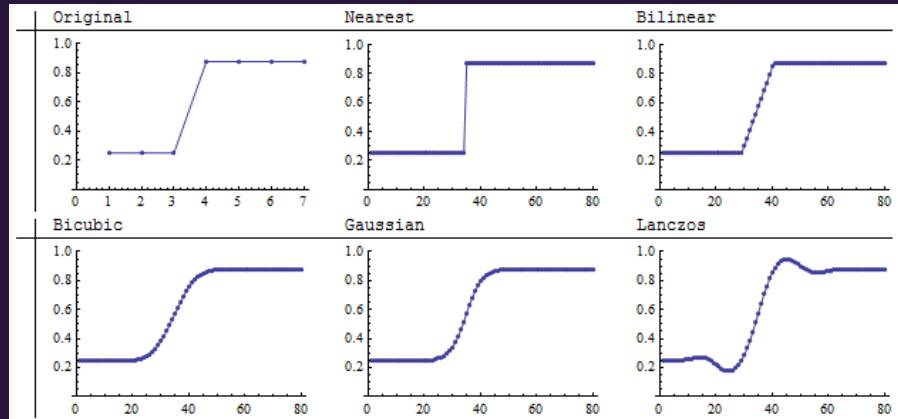


we used the Lanczos resize method



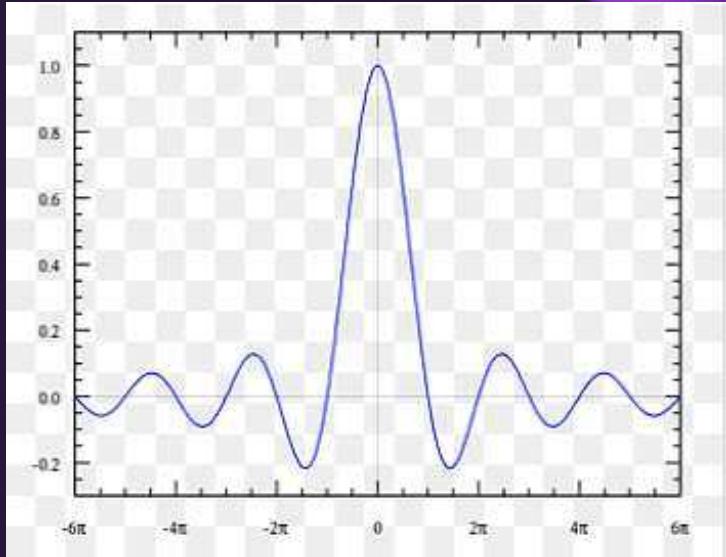


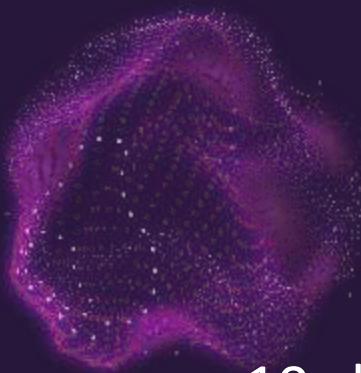
Lanczos in contrast to other interpolation methods



How does Lanczos work?

Lanczos resampling uses the sinc function to blend nearby pixel values based on their distances from the target pixel. This weighted averaging process ensures that fine details are preserved and artifacts are minimized. The resulting downscaled image aims to maintain image quality while reducing its size.





Our dataset consists of:



Cifar10:

10 classes of 5,000
images

Cifar100:

3 classes of 2,500
images



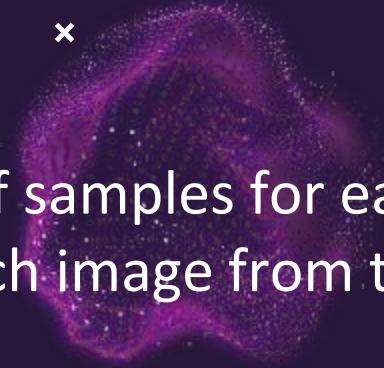
The data is
unbalanced!



DATA AUGMENTATION

THE SOLUTION:

Doubling the number of samples for each class from Cifar100 by rotating each image from the dataset



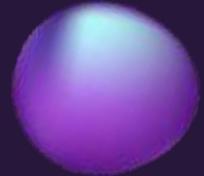
THE RESULT:

A completely balanced data set.



Data augmentation

Data augmentation involves applying various transformations and modifications to the existing dataset to create new instances of data. The goal is to increase the diversity of the dataset and help the model generalize better



Data augmentation offers several advantages in machine learning



Enhanced Generalization



Better handling of unexpected changes and noise



Reduced Overfitting



Addressing Imbalance



Optimal Resource Use



Better Results



Data format:

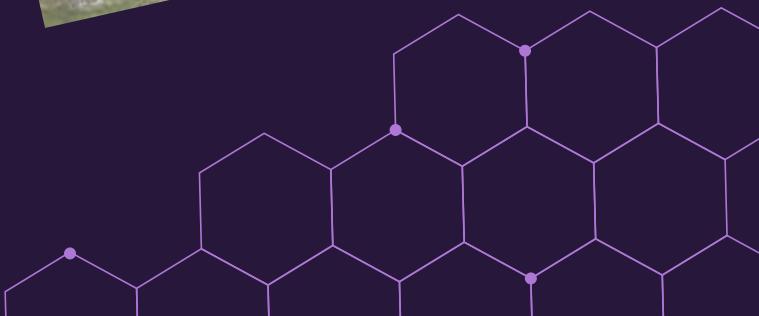
Npy files

Which have many advantages:

- + Fast Readability
- + Portability
- + Memory Efficiency
- + Supports Training Process
- + Customization

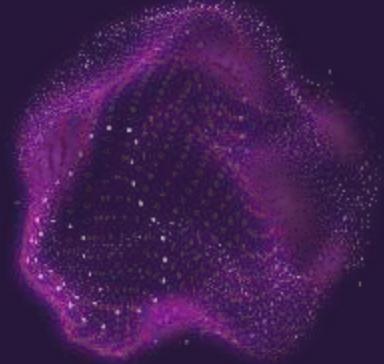
We have one problem:

- + The Cifar10 dataset is old (since 2008)
The cars, airplane, trucks and ships look different nowadays.
- + our data includes only 60,000 images for ten categories which is not much for very accurate predictions





The solution:



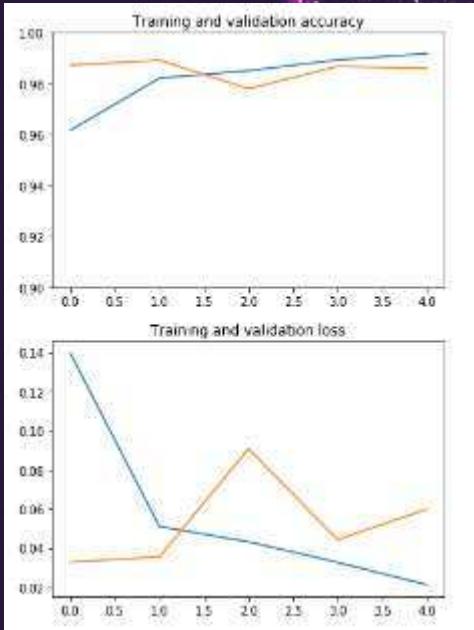
Adding a feature that allows users to upload their own images, and our system, powered by a pre-trained ResNet50 model which is based on the large imangenet dataset, classifies these images into predefined categories



Why use resnet50?

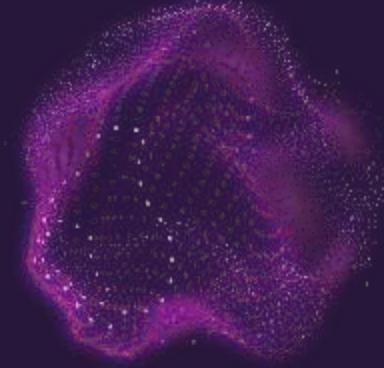
ResNet-50 is a powerful deep learning model known for its:

- 50-layer deep neural network architecture.
- Pre-training on the extensive ImageNet dataset for feature learning.
- Versatility for transfer learning across various computer vision tasks.
- excels in image classification and is widely used in research and real-world applications because of its accuracy and adaptability





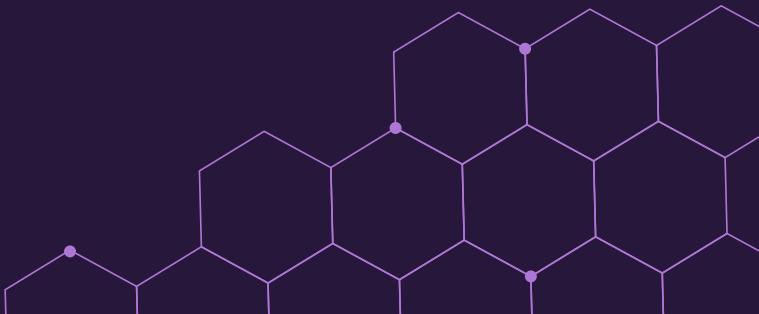
In Summary:



The ResNet-50 model is well-suited for preprocessing due to its high accuracy

it's more practical to employ a simpler model for the actual classification task

primarily because it demands extensive computational resources.



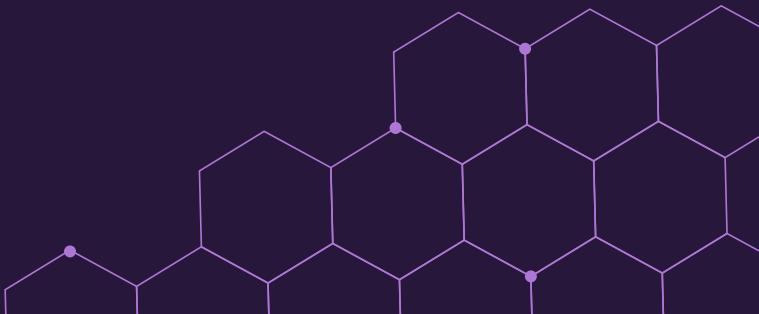
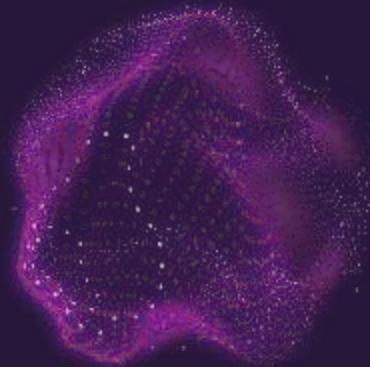
Almost done... but oops one more problem!

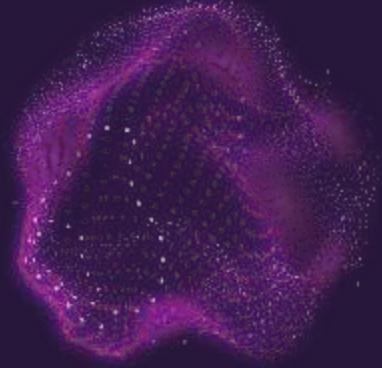


The imagenet dataset
has 1,000 classes and
cifar10 has only 10



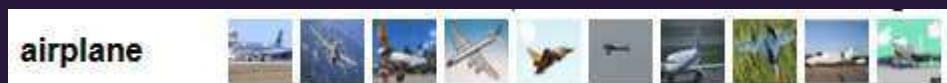
Many classes in imagenet
are subclasses of classes in
cifar10





For example:

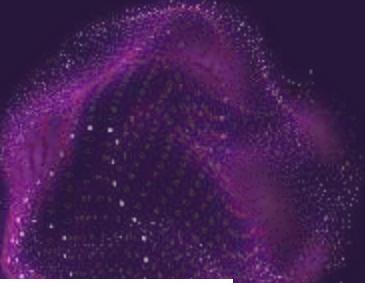
In cifar10 there is an airplane class



But in imagenet there are numerous different classes of different kinds of airplanes

narrowbody aircraft, narrow-body aircraft, narrow-body
widebody aircraft, wide-body aircraft, wide-body, twin-aisle airplane





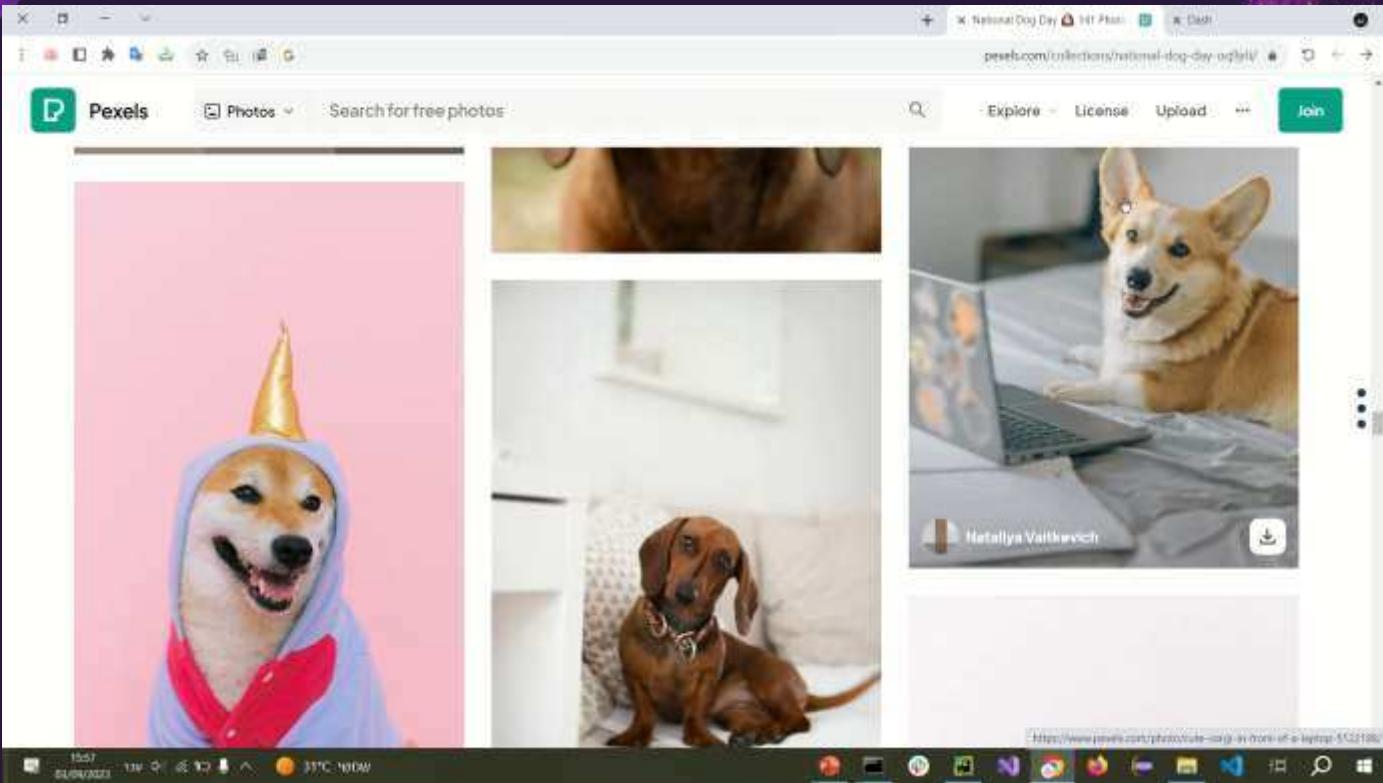
The solution:

A python script that gets a txt file in the following format and maps the imagenet classes to match the cifar10 classes

```
n03174079: delta wing
n03335030: fighter, fighter aircraft, attack aircraft
---n03577672: interceptor
---n03608074: kamikaze
---n04308397: stealth fighter
n03490784: hangar queen
n03595860: jet, jet plane, jet-propelled plane
---n03321419: fanjet, fan-jet, turbofan, turbojet
n03596543: jetliner
n03604311: jumbojet, jumbo jet
n04503499: twinjet
n03783873: monoplane
n03798610: multiengine airplane, multiengine plane
n04012084: propeller plane
```



Our GUI

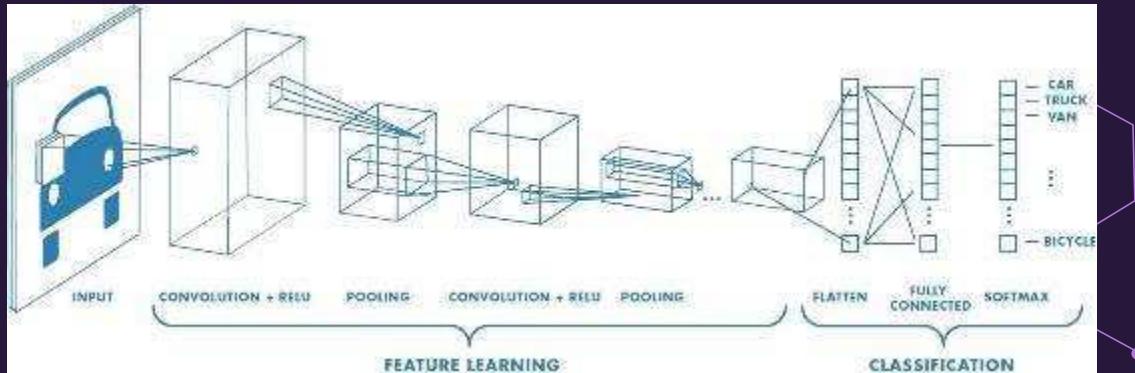


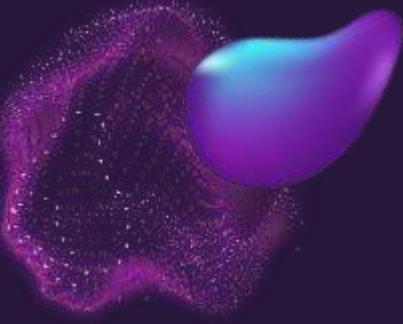
GENERAL EXPLANATION ABOUT CNN (NOGAH GROUP)



CNN MAIN LAYERS

- Convolution layer
- ReLU layer
- Dropout Layer
- Max Pooling Layer
- Fully Connected Layer
- SoftMax Layer

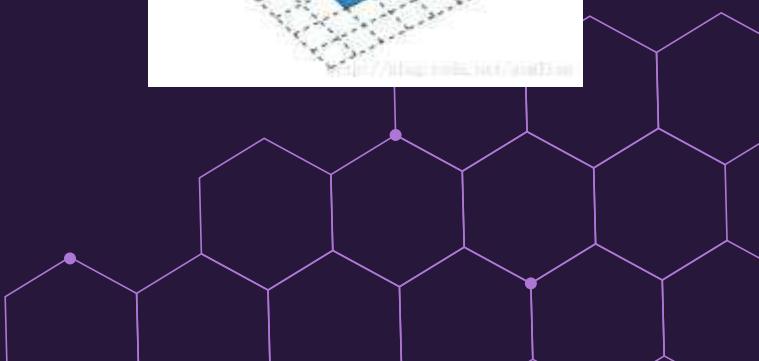
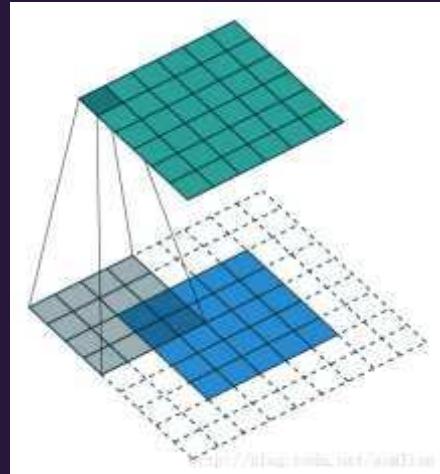




CONVOLUTION LAYER

A convolutional layer is the main building block of a CNN. It contains a set of filters (or kernels), parameters of which are to be learned throughout the training.

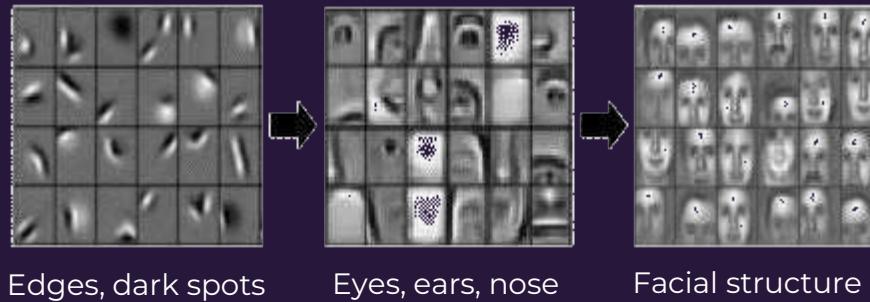
Convolutional layers apply a convolution operation to the input, passing the result to the next layer. A convolution converts all the pixels in its receptive field into a single value.



CONVOLUTION KERNELS

The activation maps output by convolutional layers can be inspected to understand exactly what features detected for a given input image.

Visualizing the filters within a learned convolutional neural network can provide insight into how the model works.

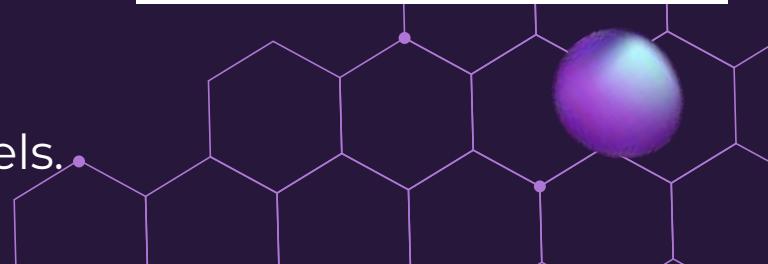
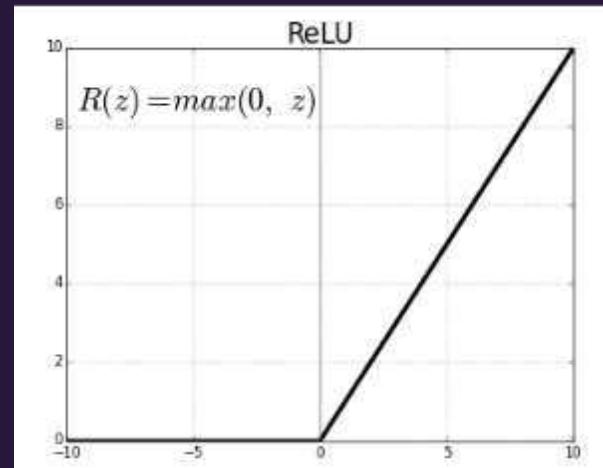


RELU LAYER

After each convolution operation, a CNN applies a Rectified Linear Unit (ReLU) transformation to the feature map, introducing nonlinearity to the model.

The purpose of applying the ReLU function is to increase the non-linearity in our images, in order to make up for the linearity that we might impose on an image when we put it through the convolution operation.

The ReLU is the most commonly used activation function in deep learning models.

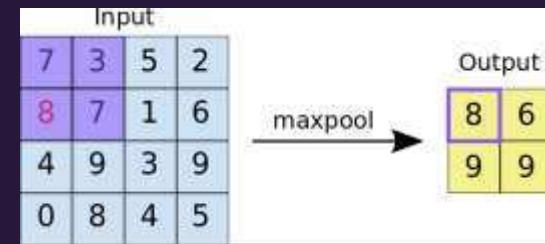


MAX POOLING LAYER

Pooling layers are used to reduce the dimensions of the feature maps. It summarizes the features present in a region of the feature map generated by a convolution layer. So further operations are performed on summarized features instead of precisely positioned features generated by the convolution layer.

This makes the model more robust to variations in the position of the features in the input image.

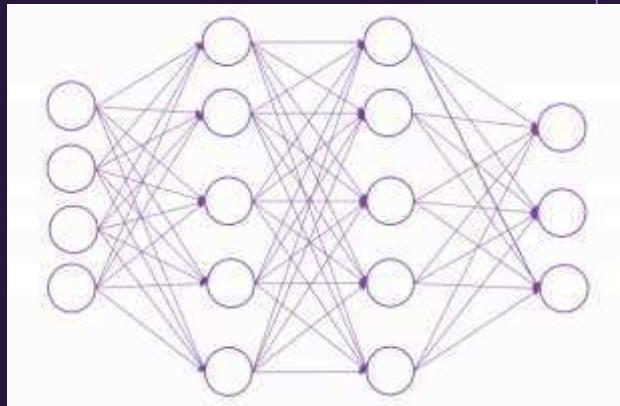
The output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.



DROPOUT LAYER

Dropout is a regularization technique for neural network models, where randomly selected neurons are ignored during training. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass, and any weight updates are not applied to the neuron on the backward pass.

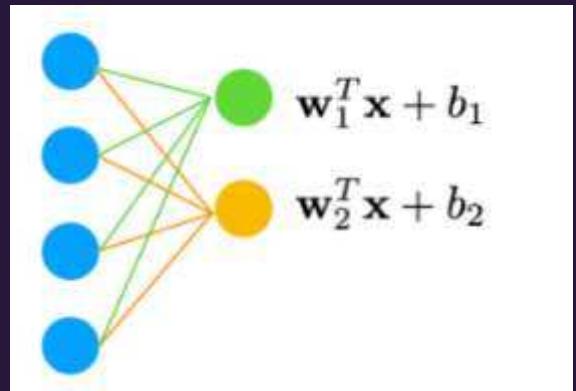
The effect is that the network becomes less sensitive to the specific weights of neurons. This, in turn, results in a network capable of better generalization and less likely to overfit the training data.



FULLY CONNECTED LAYER

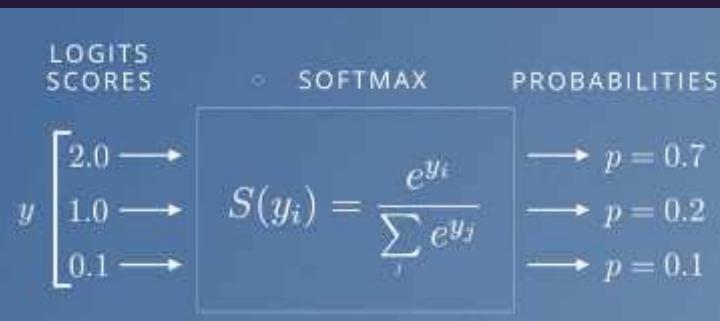
A fully connected layer in neural network is where all the inputs from one layer are connected to every activation unit of the next layer. As a result, all possible connections layer-to-layer are present, meaning every input of the input vector influences every output of the output vector. However, not all weights affect all outputs.

The Fully connected layer is used for classifying the input image into a label. It connects the information extracted from the previous steps to the output layer and eventually classifies the input into the desired label.



SOFTMAX LAYER

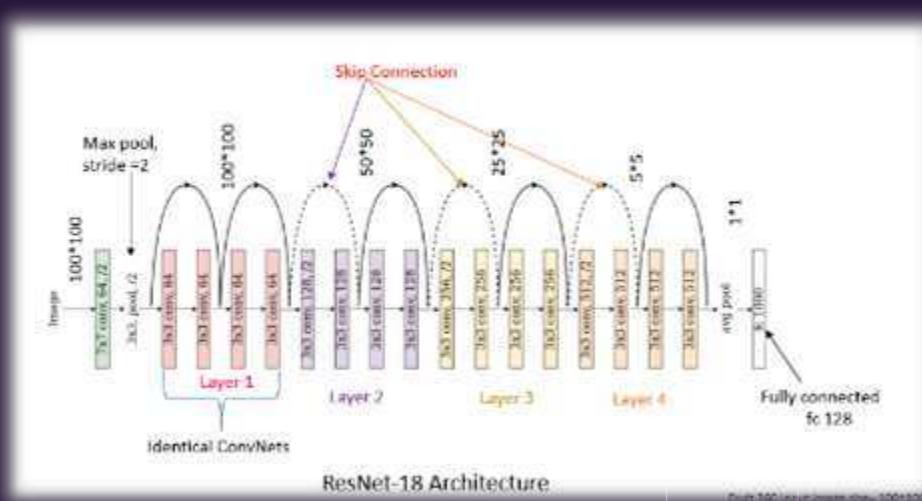
The Softmax function is a crucial component in the output layer of CNNs for classification tasks. It transforms the raw scores produced by the network into a meaningful probability distribution over classes, making it easier to interpret and use the model's predictions.



RESNET-18

Residual Network (ResNet) is a deep learning model used for computer vision applications. It is a Convolutional Neural Network (CNN) architecture designed to support hundreds or thousands of convolutional layers.

Previous CNN architectures were not able to scale to a large number of layers, which resulted in limited performance. However, when adding more layers, researchers faced the “vanishing gradient” problem.



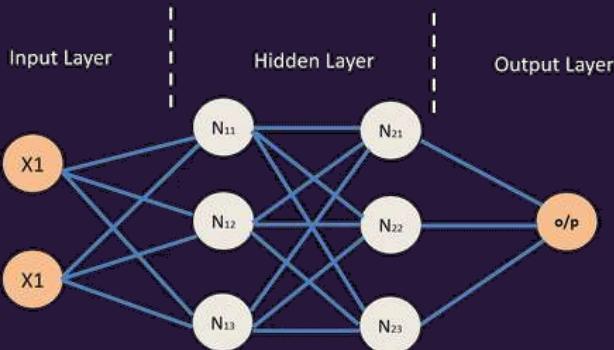
BACKPROPAGATION



BACKPROPAGATION

Backpropagation is a fundamental technique used in training neural networks. It involves the iterative adjustment of weights within a network to minimize errors and enhance its ability to map inputs to outputs accurately. By comprehending the essence of backpropagation, we can gain insights into how neural networks learn and improve their predictions.

Neural Network – Backpropagation

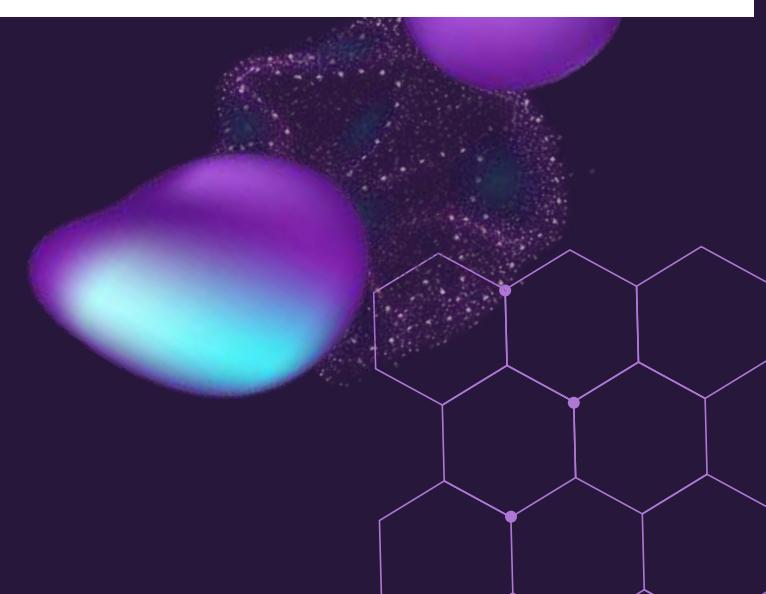




THE IMPLEMENTATION

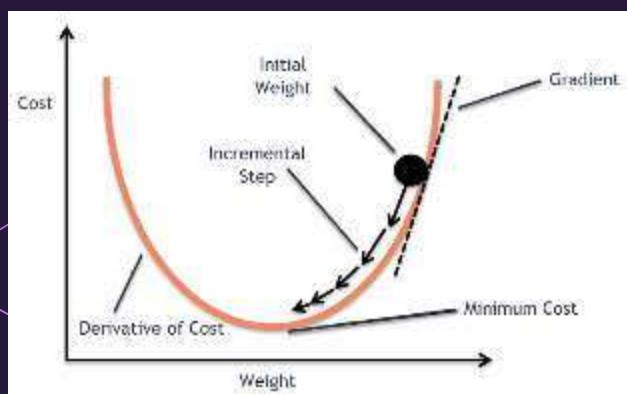
To implement backpropagation, the chain rule of differentiation plays a pivotal role. This mathematical concept allows us to efficiently compute derivatives of the loss function with respect to the network's weights, particularly in the context of a neural network's layered architecture. The chain rule unravels how changes in weights influence the overall loss, providing crucial insights into how to minimize it. This intricate process serves as the foundation for refining the network's weights and improving its performance.

$$\begin{aligned}\frac{dy}{dx} &= \frac{dy}{du} \frac{du}{dx} \\ \frac{d^2y}{dx^2} &= \frac{d^2y}{du^2} \left(\frac{du}{dx} \right)^2 + \frac{dy}{du} \frac{d^2u}{dx^2} \\ \frac{d^3y}{dx^3} &= \frac{d^3y}{du^3} \left(\frac{du}{dx} \right)^3 + 3 \frac{d^2y}{du^2} \frac{du}{dx} \frac{d^2u}{dx^2} + \frac{dy}{du} \frac{d^3u}{dx^3} \\ \frac{d^4y}{dx^4} &= \frac{d^4y}{du^4} \left(\frac{du}{dx} \right)^4 + 6 \frac{d^3y}{du^3} \left(\frac{du}{dx} \right)^2 \frac{d^2u}{dx^2} + \frac{d^2y}{du^2} \left(4 \frac{du}{dx} \frac{d^3u}{dx^3} + 3 \left(\frac{d^2u}{dx^2} \right)^2 \right) + \frac{dy}{du} \frac{d^4u}{dx^4}.\end{aligned}$$



GRADIENT DESCENT

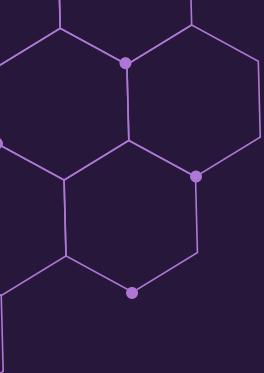
Gradient Descent (GD) is a commonly used optimization technique that leverages backpropagation. GD updates the weights by analyzing the gradients of the loss function with respect to the weights. Through backpropagation, GD identifies which weights should be increased or decreased to minimize the loss. This process is performed in a reverse order, as adjustments to earlier layers affect subsequent ones due to the interconnected nature of neural networks.



Model Construction and Training Process **(Bereshit GROUP)**



BOOTCAMP, 2023



x

STAGES:



x

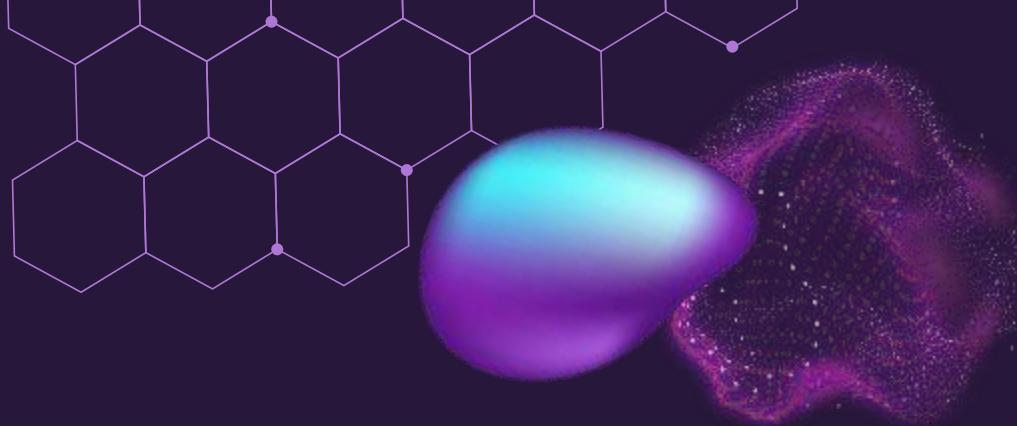
01 **The network structure**

02 **Training process**

03 **Out of Distribution**

01

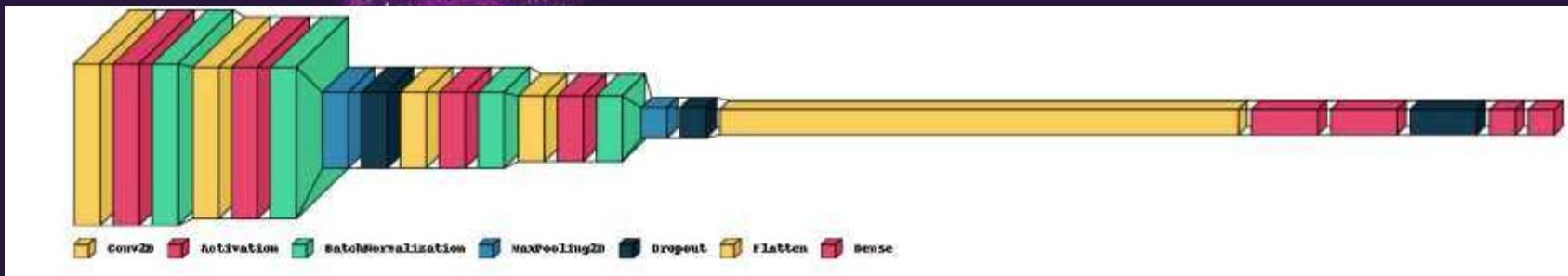
Network structure



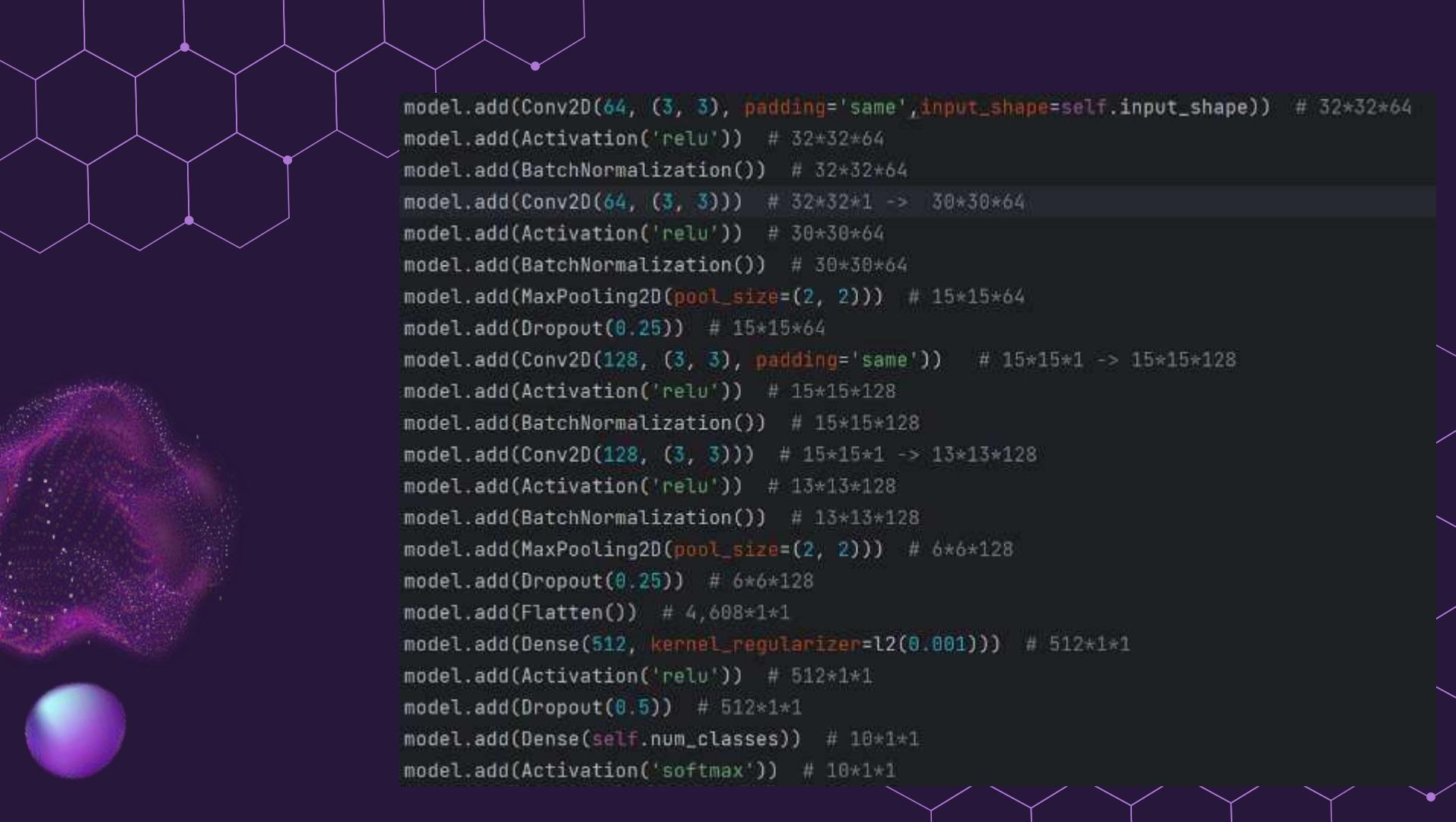
+



Model architecture



Conv2D->Relu->BatchNormalization->Conv2D->Relu->BatchNormalization->MaxPooling2D->Dropout->Conv2D->
Relu-> BatchNormalization->Conv2D->Relu->BatchNormalization->MaxPooling2D->Dropout->Flatten->Dense->Relu->Dropout-> Dense->SoftMax



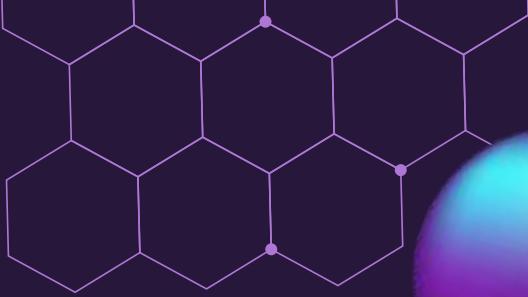
```
model.add(Conv2D(64, (3, 3), padding='same', input_shape=self.input_shape)) # 32*32*64
model.add(Activation('relu')) # 32*32*64
model.add(BatchNormalization()) # 32*32*64
model.add(Conv2D(64, (3, 3))) # 32*32*1 -> 30*30*64
model.add(Activation('relu')) # 30*30*64
model.add(BatchNormalization()) # 30*30*64
model.add(MaxPooling2D(pool_size=(2, 2))) # 15*15*64
model.add(Dropout(0.25)) # 15*15*64
model.add(Conv2D(128, (3, 3), padding='same')) # 15*15*1 -> 15*15*128
model.add(Activation('relu')) # 15*15*128
model.add(BatchNormalization()) # 15*15*128
model.add(Conv2D(128, (3, 3))) # 15*15*1 -> 13*13*128
model.add(Activation('relu')) # 13*13*128
model.add(BatchNormalization()) # 13*13*128
model.add(MaxPooling2D(pool_size=(2, 2))) # 6*6*128
model.add(Dropout(0.25)) # 6*6*128
model.add(Flatten()) # 4,608*1*1
model.add(Dense(512, kernel_regularizer=l2(0.001))) # 512*1*1
model.add(Activation('relu')) # 512*1*1
model.add(Dropout(0.5)) # 512*1*1
model.add(Dense(self.num_classes)) # 10*1*1
model.add(Activation('softmax')) # 10*1*1
```

02

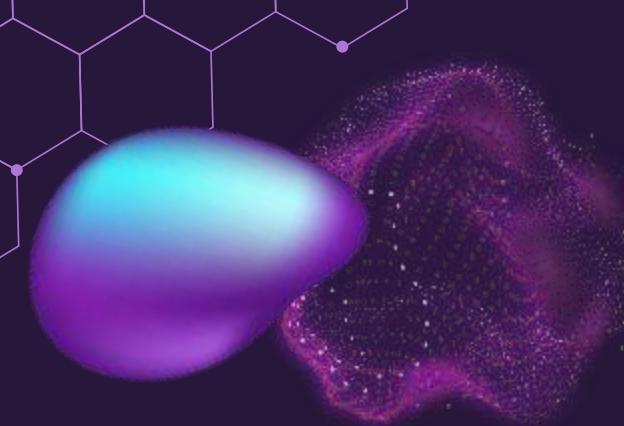
Training process



x



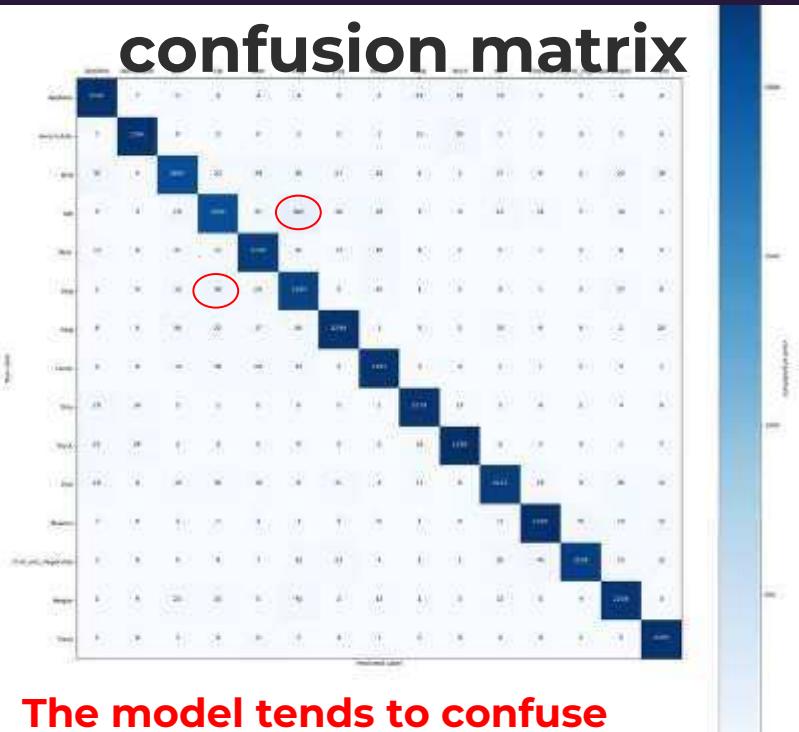
+



x

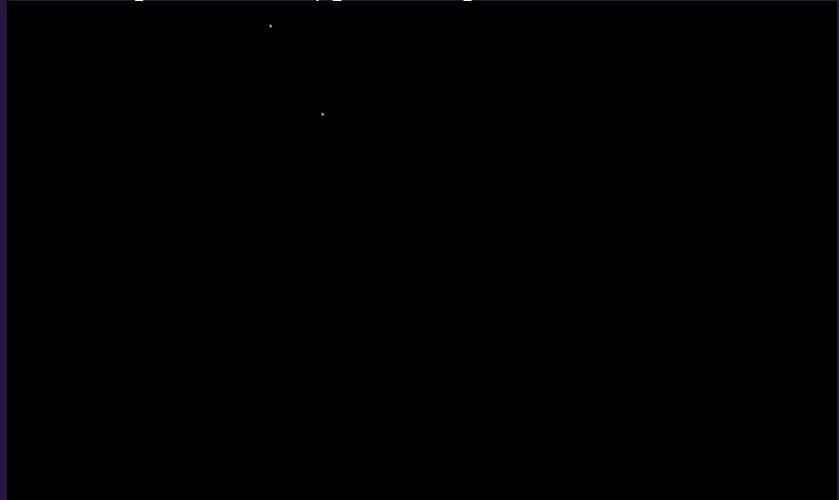


Visualization



The model tends to confuse
dogs vs. cats

printing of epochs



The final loss and accuracy

Test loss: 0.2915284306195472

Test accuracy: 0.780194713562619

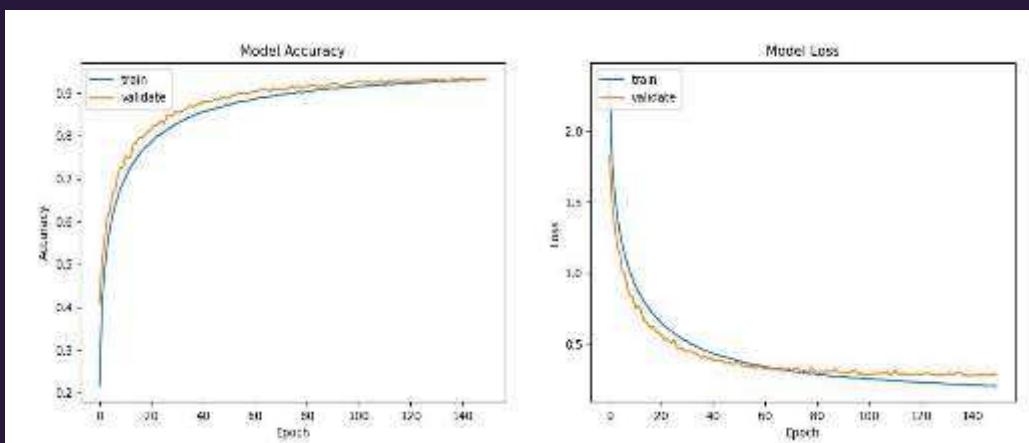


Visualization

Classification Report

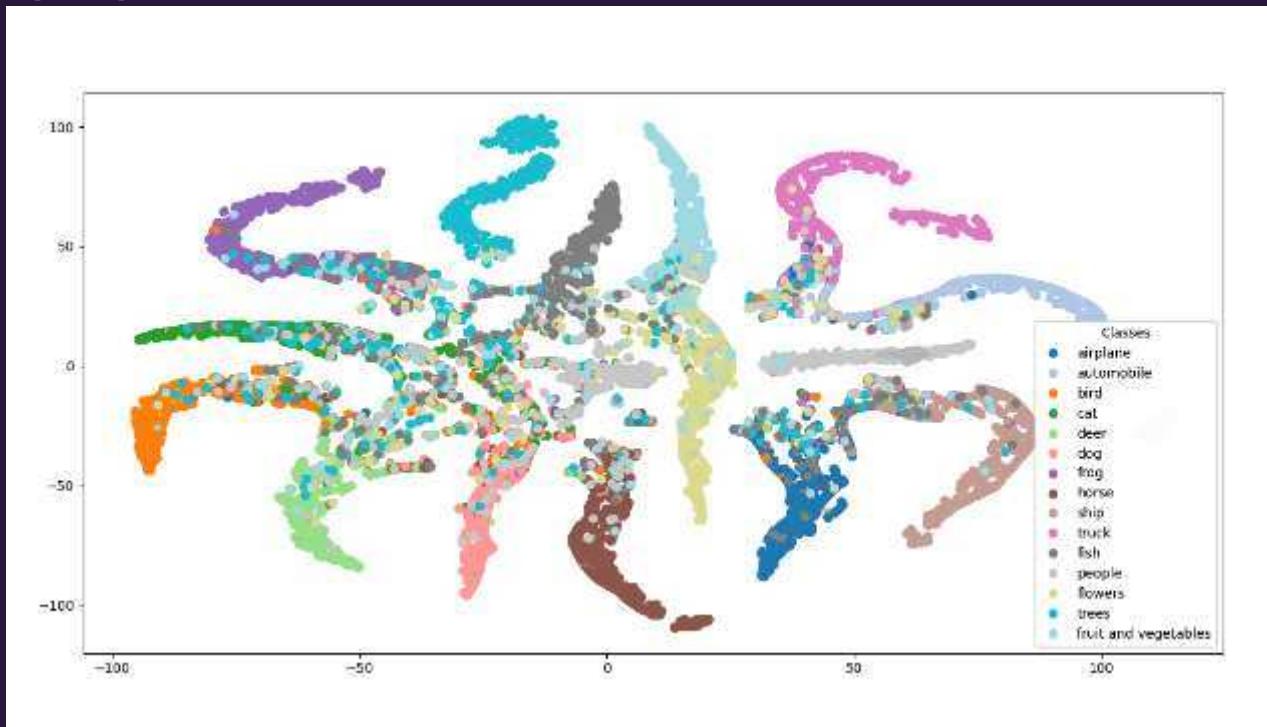
	precision	recall	f1-score	support
0	0.93	0.95	0.94	2428
1	0.97	0.96	0.97	2376
2	0.92	0.89	0.90	2327
3	0.88	0.84	0.86	2453
4	0.92	0.92	0.92	2395
5	0.86	0.91	0.88	2383
6	0.94	0.93	0.93	2473
7	0.94	0.95	0.94	2399
8	0.96	0.97	0.96	2358
9	0.94	0.97	0.95	2420
10	0.93	0.91	0.92	2436
11	0.95	0.95	0.95	2449
12	0.96	0.93	0.94	2376
13	0.93	0.94	0.93	2397
14	0.96	0.98	0.97	2330
accuracy			0.93	36000
macro avg	0.93	0.93	0.93	36000
weighted avg	0.93	0.93	0.93	36000

accuracy vs. loss



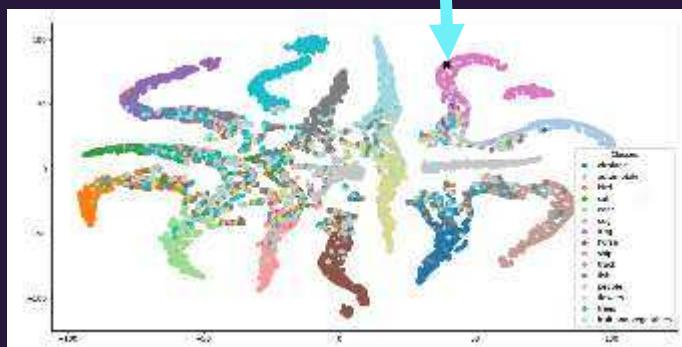
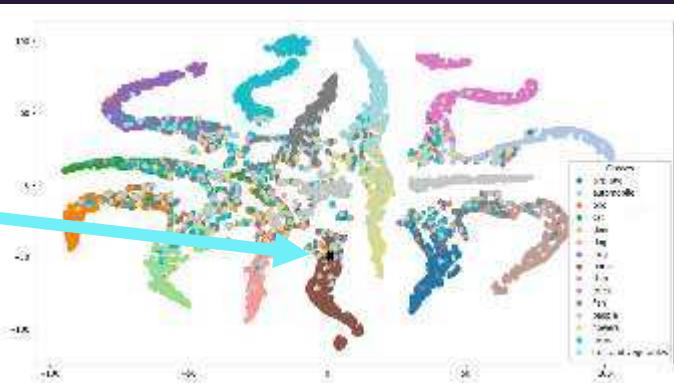
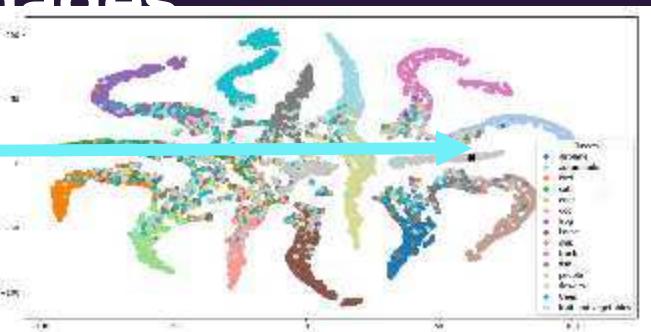
Visualization

T-sne



Visualization

T-sne new images



03

Out of Distribution



*



*

+



e.g.

Welcome to our Image Classifier

Upload Image



Sorry, your image label is not in our training set

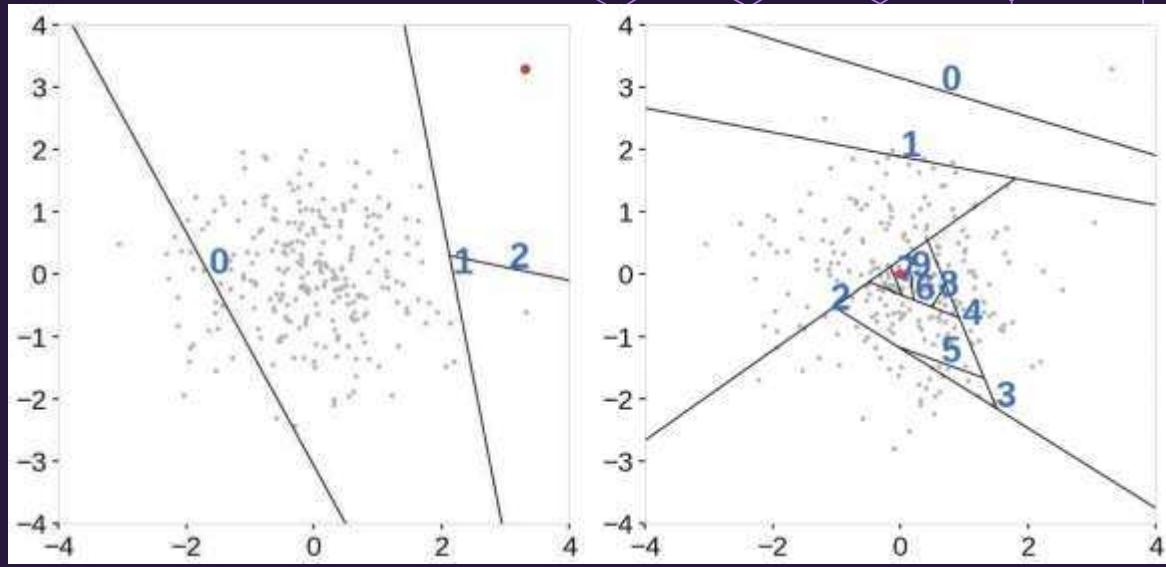


first approach: add different class: other

**positive: inlier
negative: outlier**

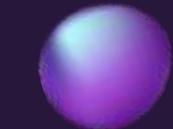


second approach: Isolation forest

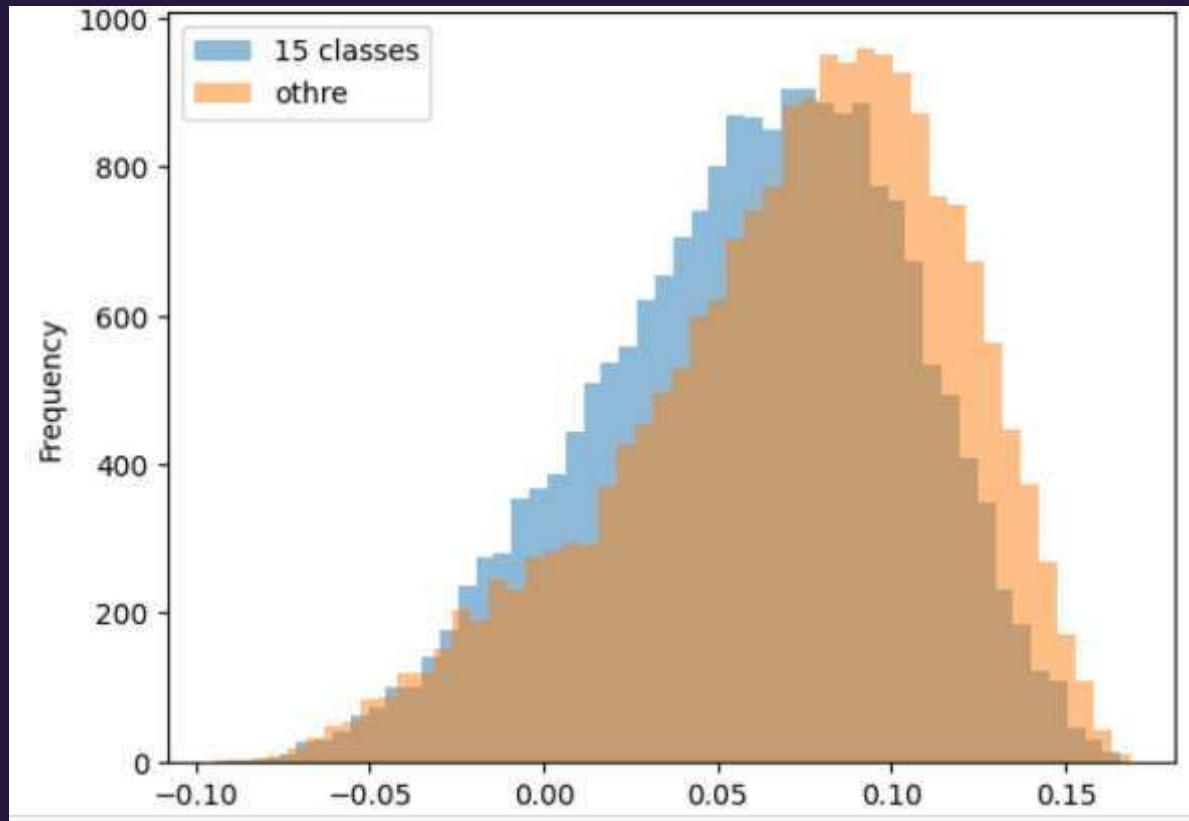


- How many splits does it take for a sample point to be isolated?
- A regular point is much harder to isolate than an anomaly point

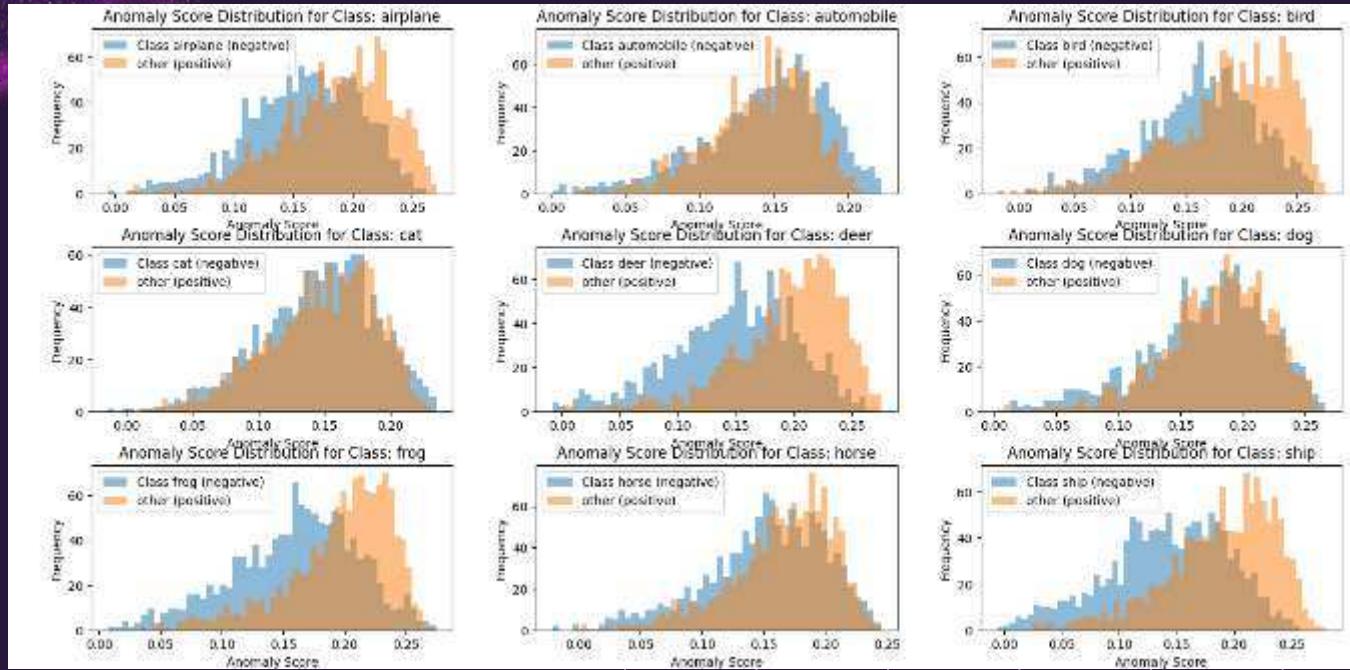
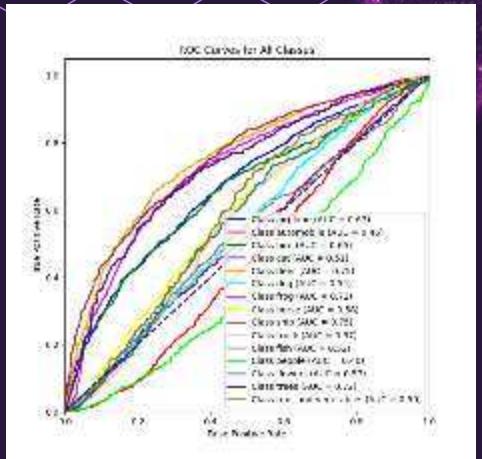
one isolation forest for all classes



positive: inlier
negative: outlier



isolation forest per class, on the image

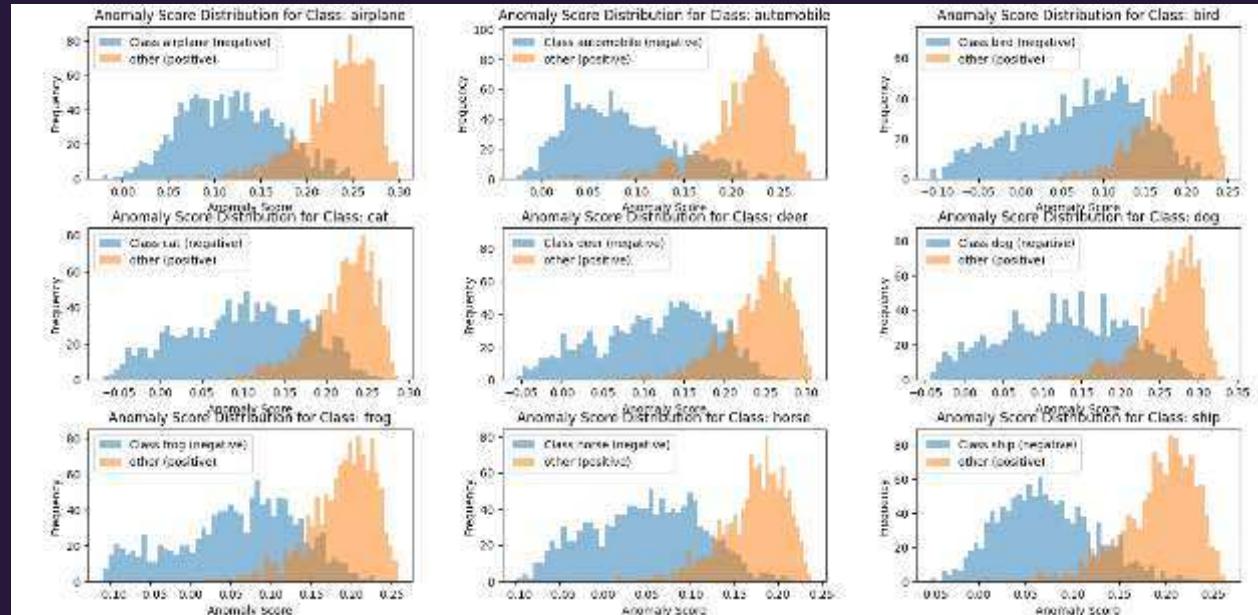


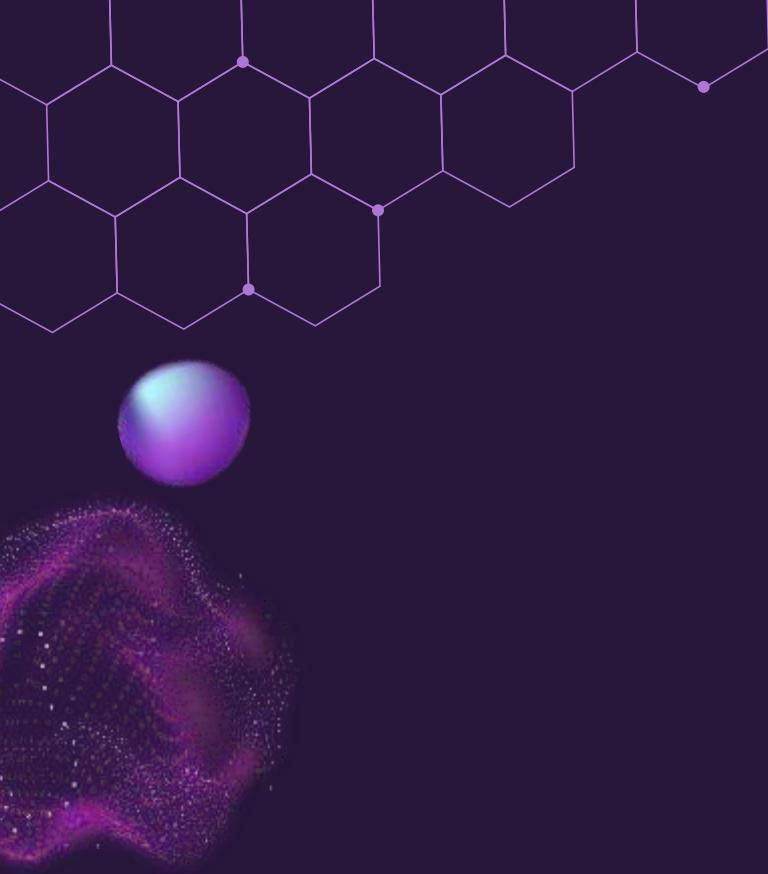
positive: inlier
negative: outlier

Classification Forest per Class, on features extracted by CNN model

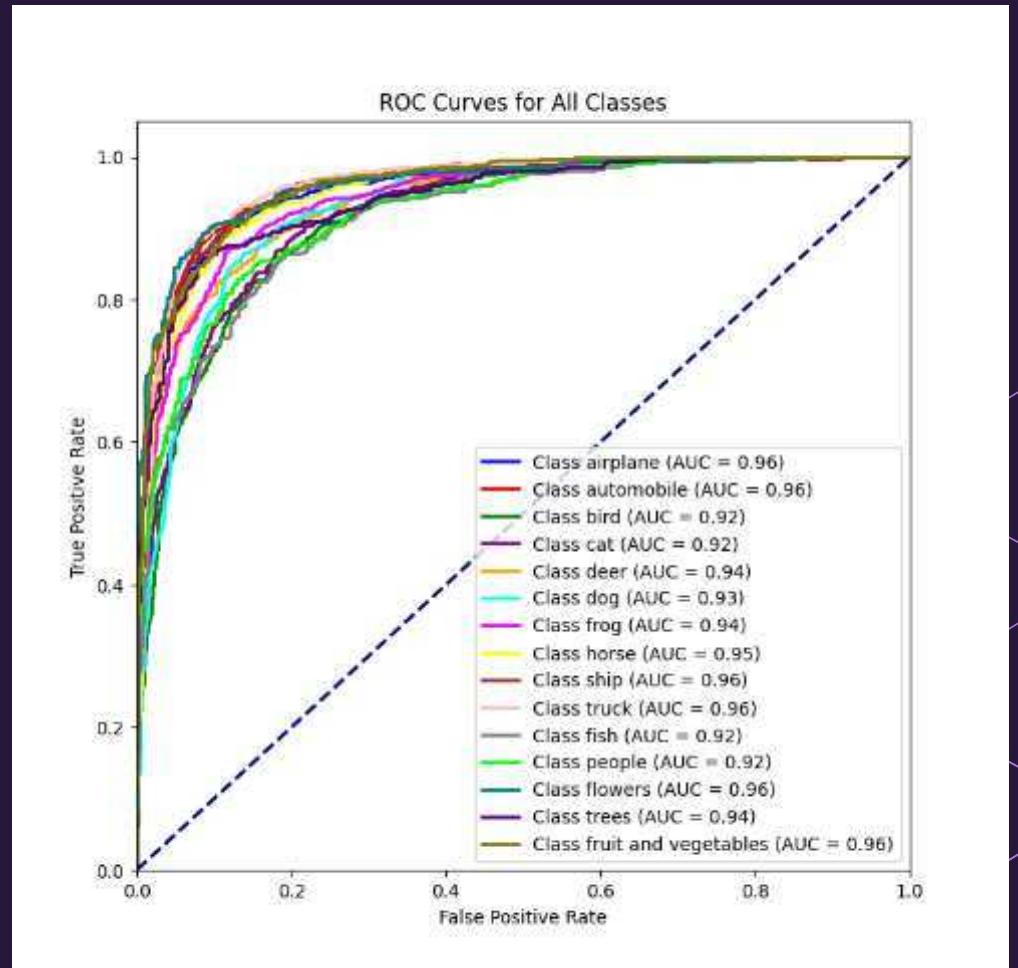


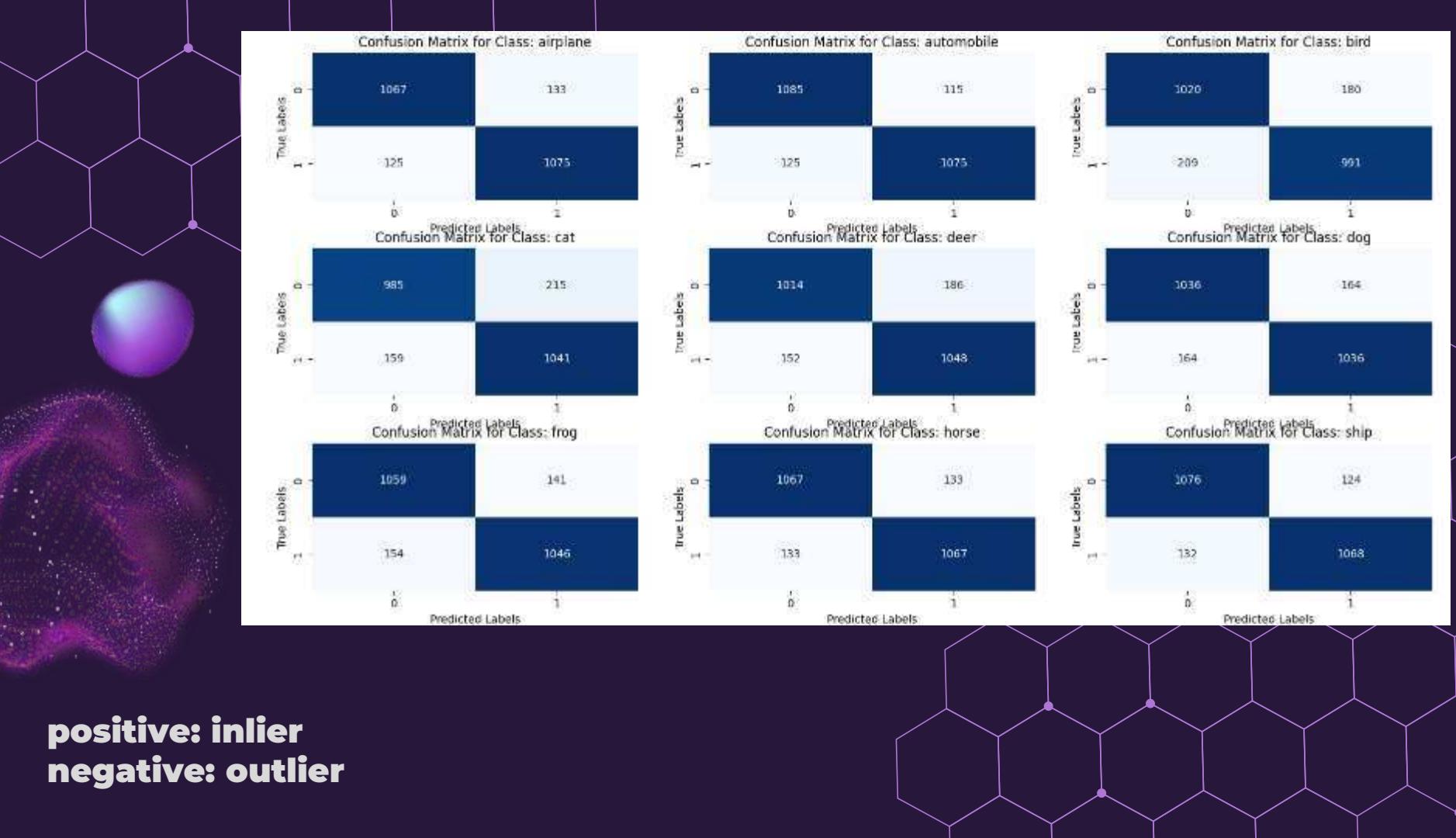
positive: inlier
negative: outlier





positive: inlier
negative: outlier





**isolation forest per class, on
features extracted by cnn
model, train on TP**



**positive: inlier
negative: outlier**



GUI

predict a label for a image with a trained label



```
loaded_model = CNN.load_cnn_model('../save_models/cnn_model_1.h5').model
probabilities = loaded_model.predict(image)
label = np.argmax(probabilities)
label_category = convert_to_category(label)
```

recognize anomalous label



```
for class_idx in range(15):
    score = models[class_idx].decision_function(features_of_image)
    if score[0] > thresholds[class_idx]:
        belongs.append(class_names[class_idx])
```

```
belongs.append(class_names[class_idx])
```



giving label based on similar image



Your image label is: airplane

Give us another chance

Explore similar images from our training set. Click the button above the image with the same label

Most Similar



Second Most Similar



Third Most Similar



Fourth Most Similar



Based on the image you selected, your label is ship

```
def find_label_of_similar_image(image_index):  
    if image_index is not None:  
        image_label = similar_images_label[image_index]  
        label = convert_to_category(image_label)  
    return label
```

finding the similar images

using distance of regular features

```
for row in features_class.values:  
    distances.append(np.sum(np.abs(row - flattened_image)))  
sorted_indices = np.argsort(distances)  
four_closest_indices = sorted_indices[:4]
```

using features after our model predict

```
model =  
    CNN.load_cnn_model('../save_models/cnn_model_1.h5').model  
feat_extractor = Model(inputs=model.model.input,  
outputs=loaded_model.model.get_layer('dense').output)  
features = feat_extractor.predict(x_test)
```

search for the similar images from potential images

```
probabilities_indices = np.argsort(probabilities.reshape(-1))  
labels = probabilities_indices[-4:]  
filtered_data = sorted_data[sorted_data['label'].isin(labels)]  
four_closest_vectors = filtered_data.iloc[:4, 2:]
```

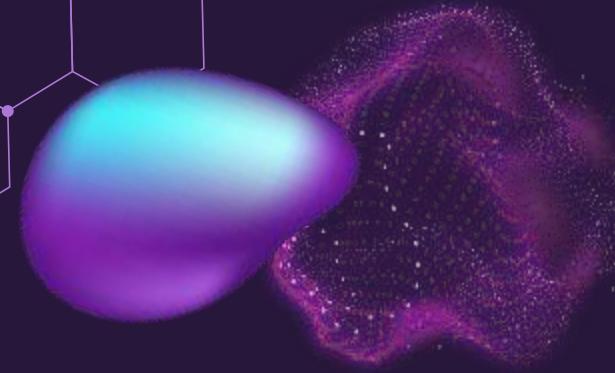
Welcome to our Image Classifier

[Upload Image](#)



04

*



DEPLOYING AND MONITORING*

NASA group:

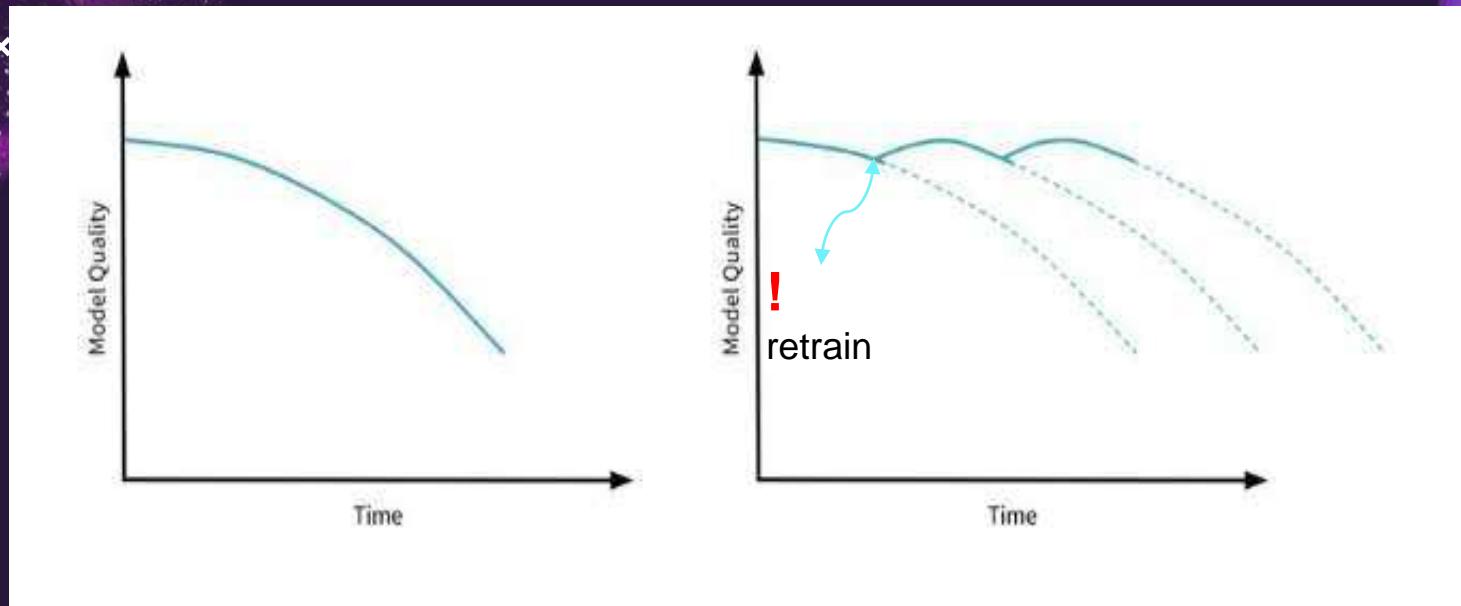
Tzofia, Esther W, Yocheved, Bracha H



*



THE GOAL:



Preventing model performance decline

* THE REASON:

**Data drift and
concept drift**



ORIGINAL MODEL



DATA DRIFT



CONCEPT DRIFT

Data Drift

x



x



Incoming data changes over time

Concept Drift:



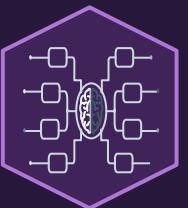
Features of target variable change over time

How to Detect Data Drift?

Monitoring metrics:

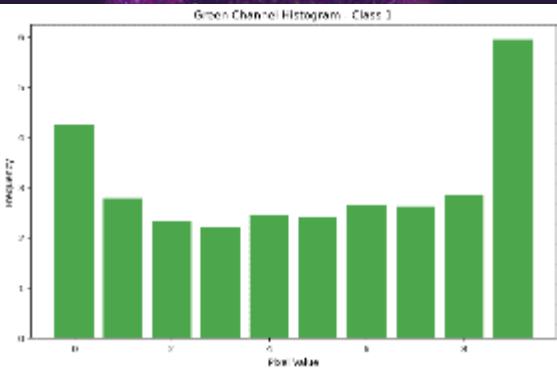
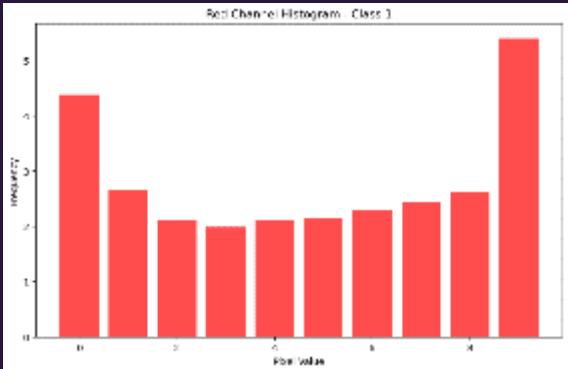


1. Distance from mean histogram of current class

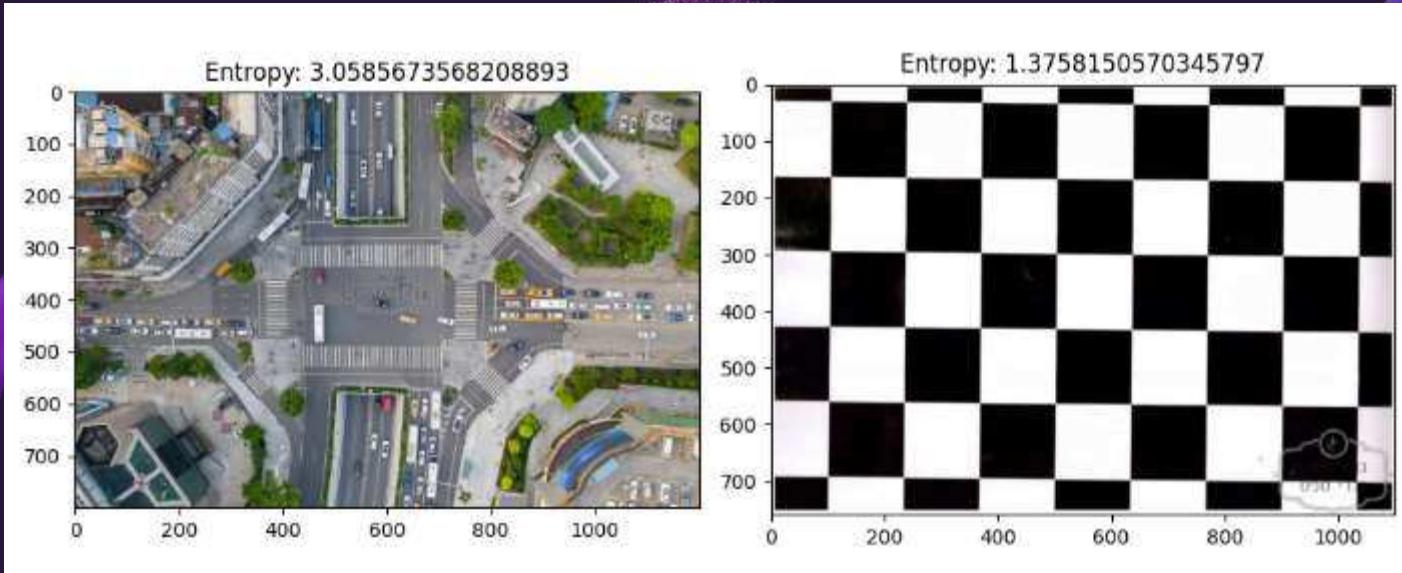


2. Entropy

What is Image Histogram?



What is Entropy?



The level of randomness or uncertainty in the pixel intensity distribution of an image, reflecting its overall complexity.

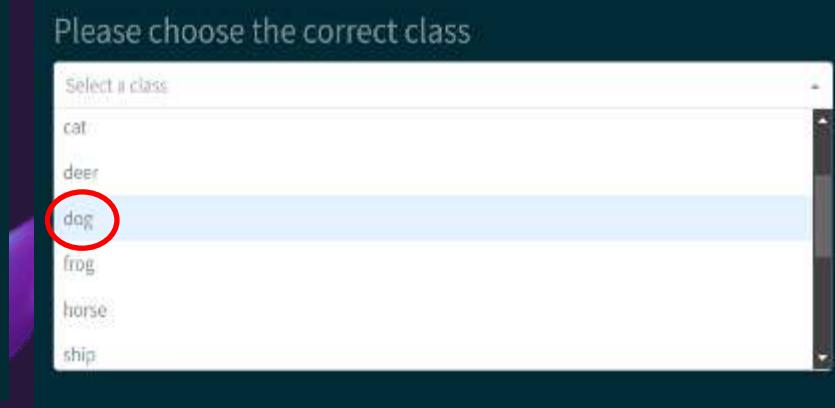
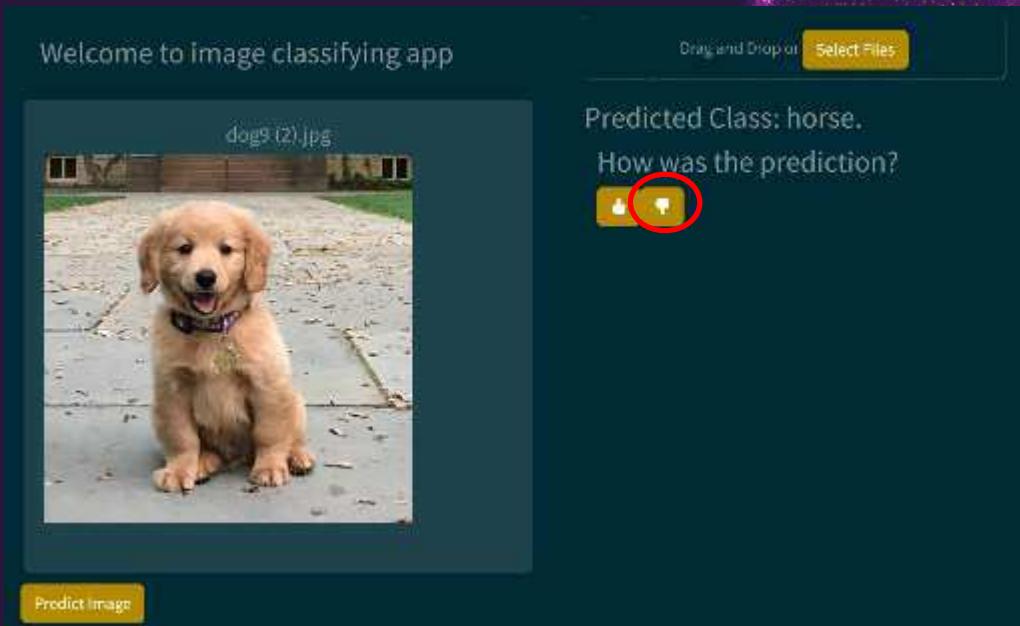
High entropy-> an image with complex patterns and visual richness.

Low entropy-> an image with simple and repetitive patterns.

How to Detect Concept Drift?

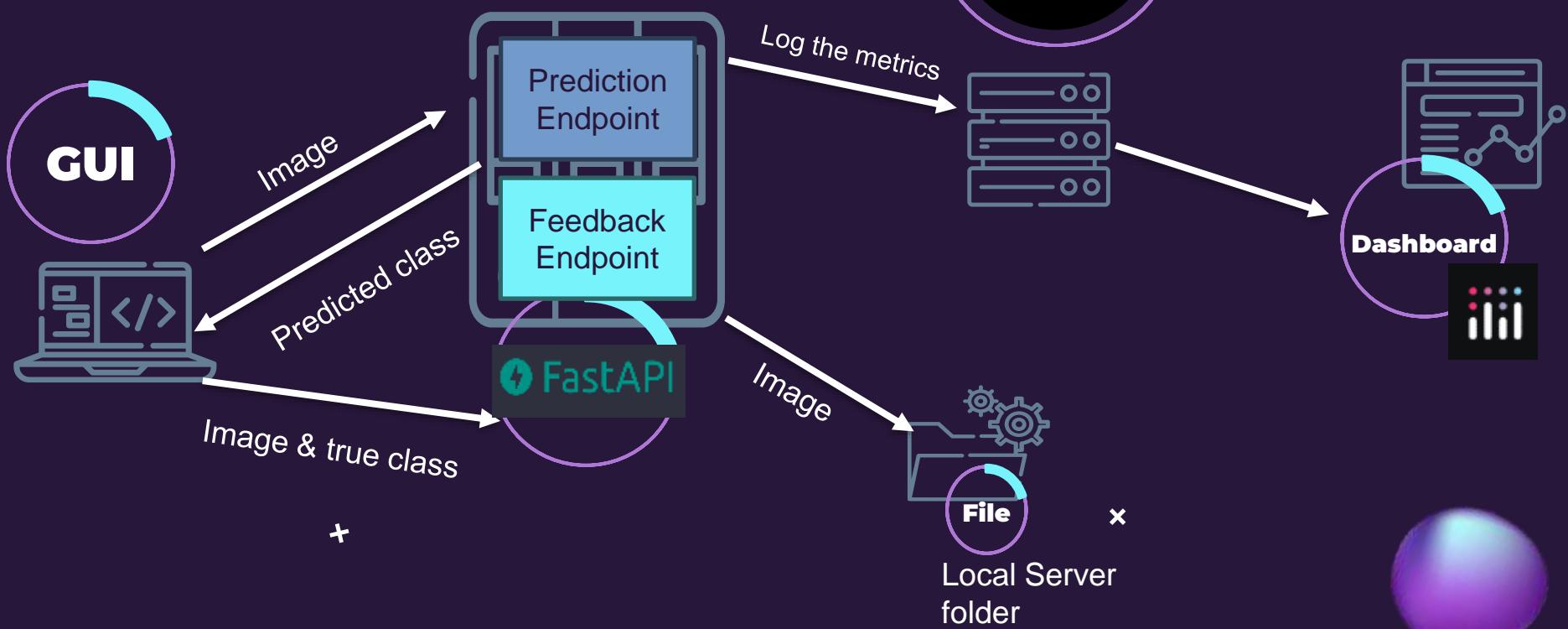
x

1. Monitoring confidence in predicted class
2. Saving wrong predicted images according to user's feedback (those images are the candidates for future training).



so... how did
we do this?

The Pipeline:





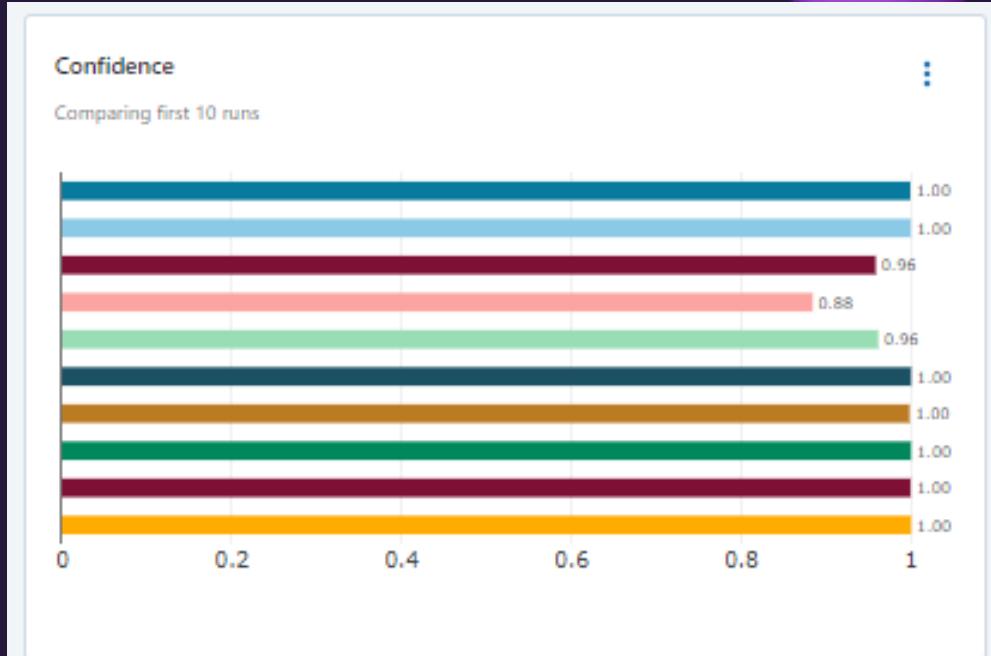
Logging metrics for each incoming image

Metrics (3)

Name	Value
Confidence	0.999
Entropy	1.871
Histogram_distance	703

Tags (1)

Name	Value
class	bird

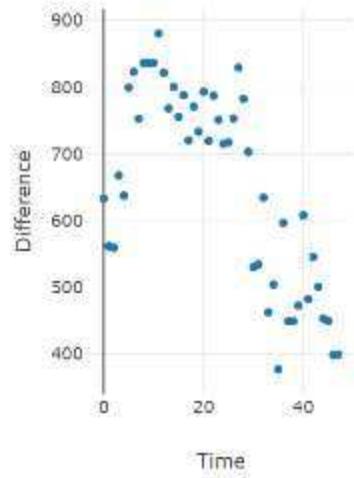


We can compare confidence of most recent 10 runs and much more!!

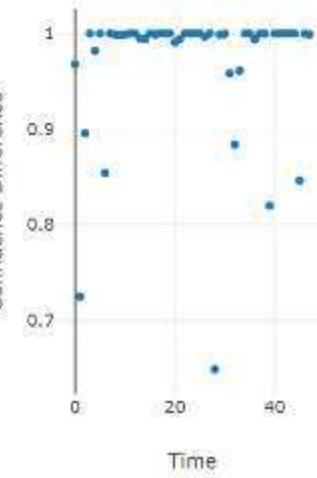
Dashboard for ML Analysis

bird

Difference from average histogram blrd



Confldence Differences for blrd



Entropy Differences for blrd

