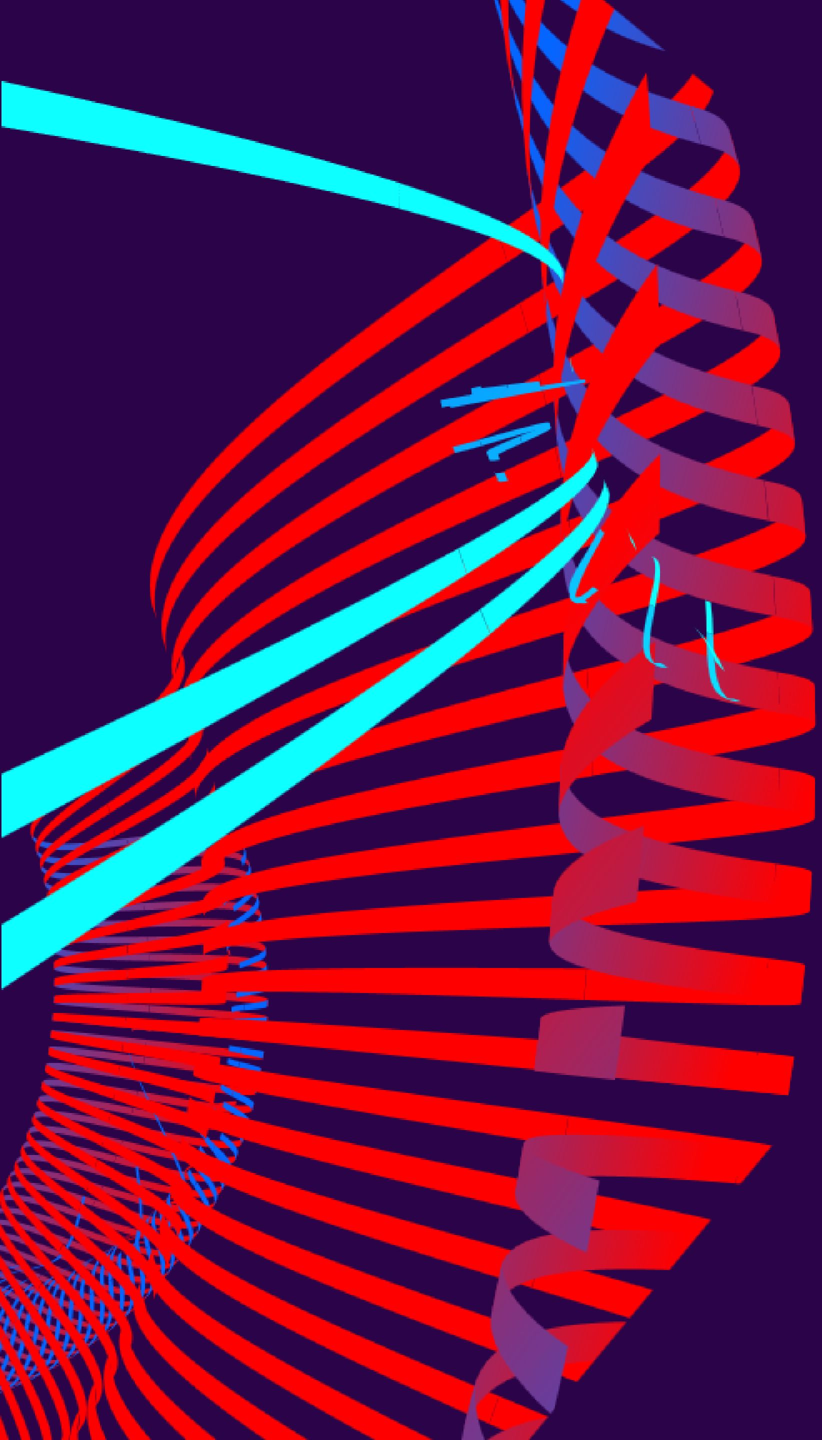
NEURAL NETWORK GRAPH CLASSIFIER

Patrick Spohr

OUTLINE



MOTIVATION

MOTIVATION

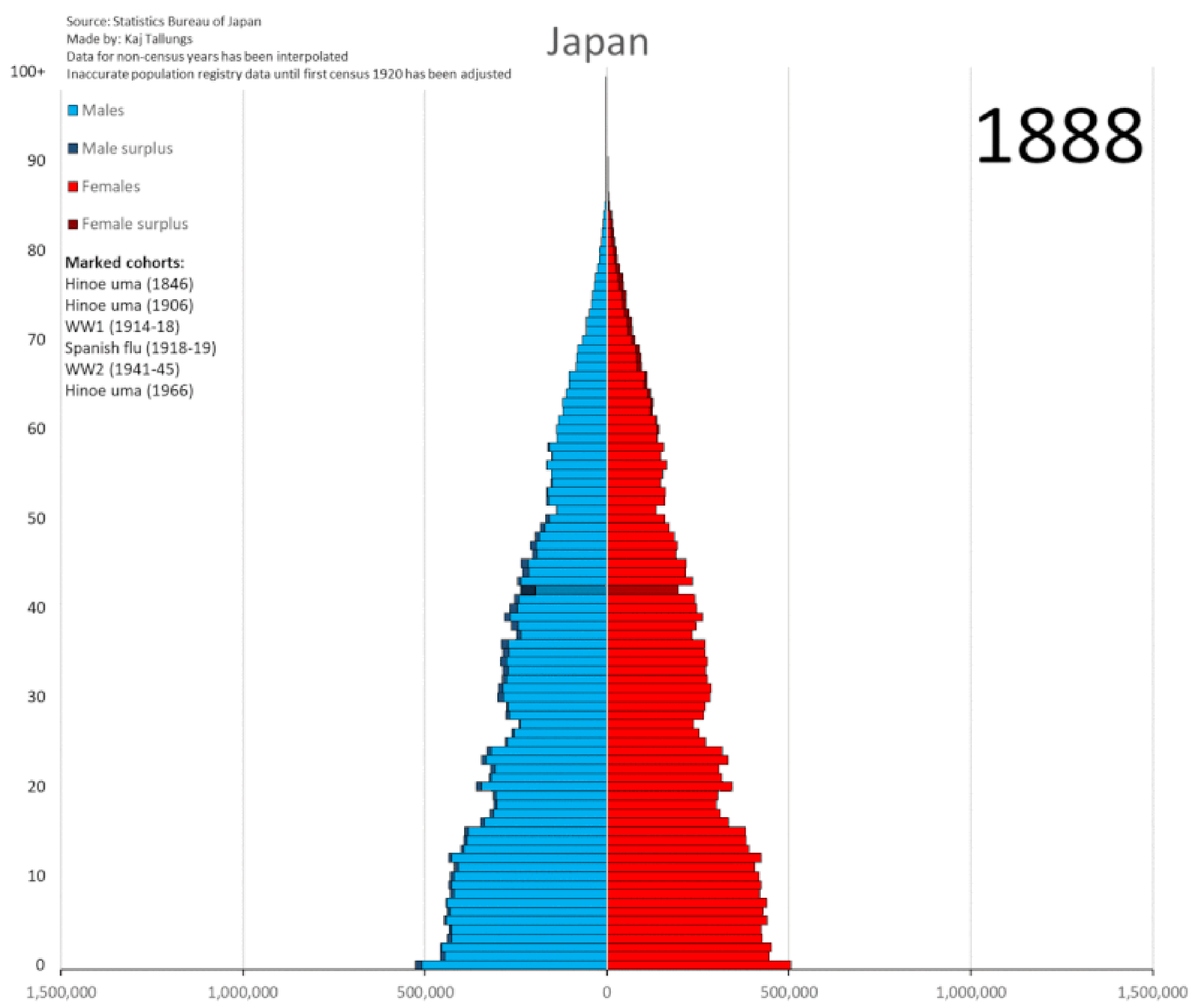
Graphs are everywhere…

And can depict many things…

MOTIVATION

I like graphs because like paintings,

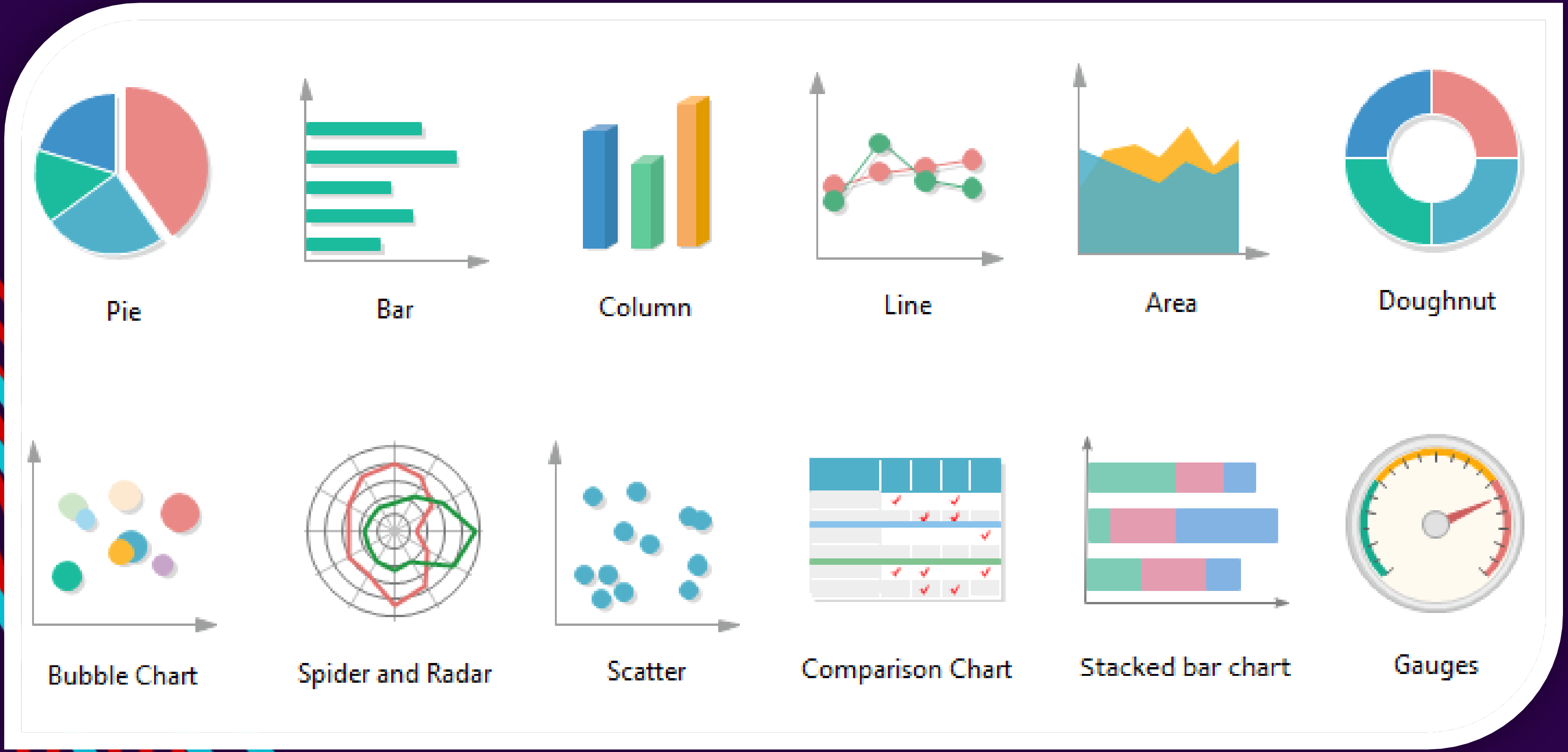
they too can tell a story.



By Kaj Tallungs - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=107393925

MOTIVATION

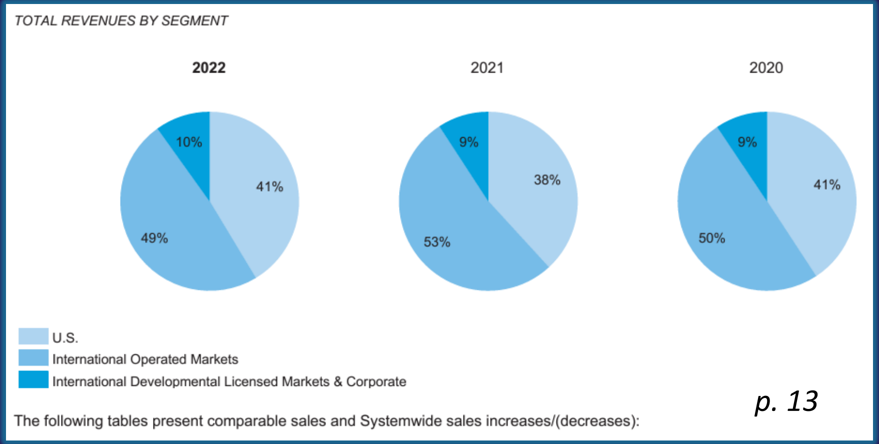
Being able to quickly identify graphs is important as there are many to choose from. *[Li]*



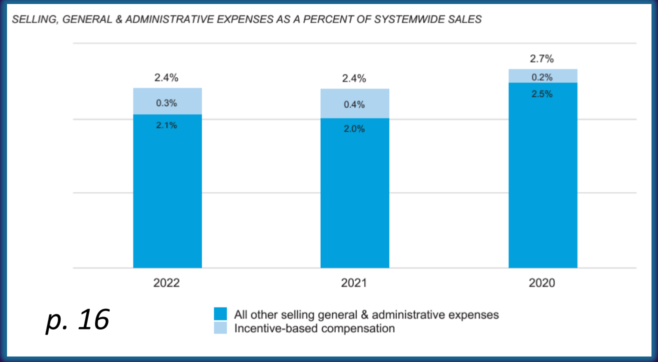
MOTIVATION

Graphs make it quick and easy to interpret a large amount of information.

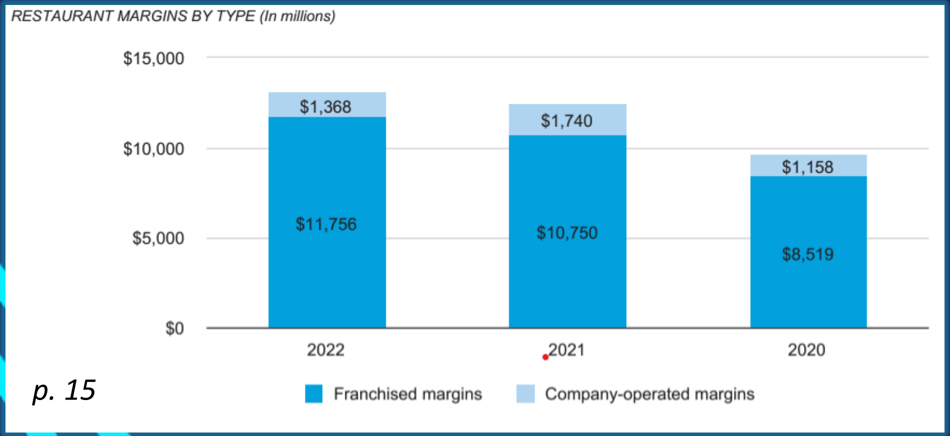
McDonald's Annual 2022 Report has 73 pages and 8 graphs. [McDonald's]



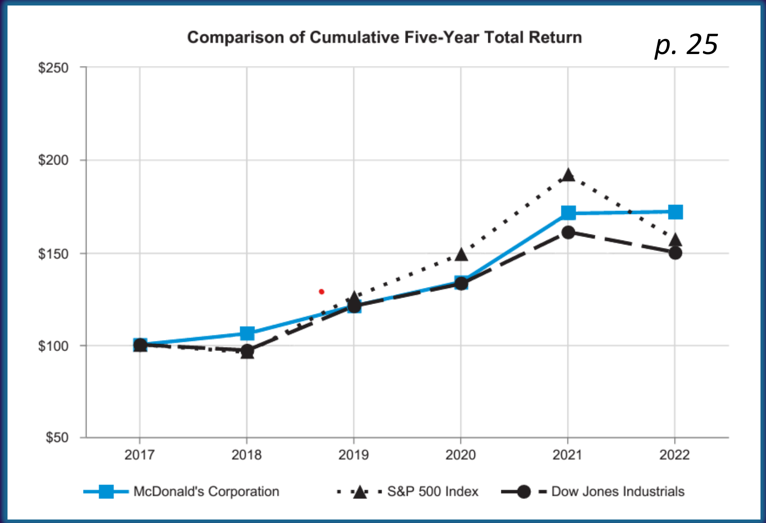
p. 13



p. 16



p. 15



p. 25

MOTIVATION

What's the possible use case of being able to identify graphs in McDonald's Annual Report?

Identify → **Locate** → **Organize**

Is it a graph?

Yes

Classify graph

Where are they?

Report

Pages 13, 15, 16, 25

What type of graph?

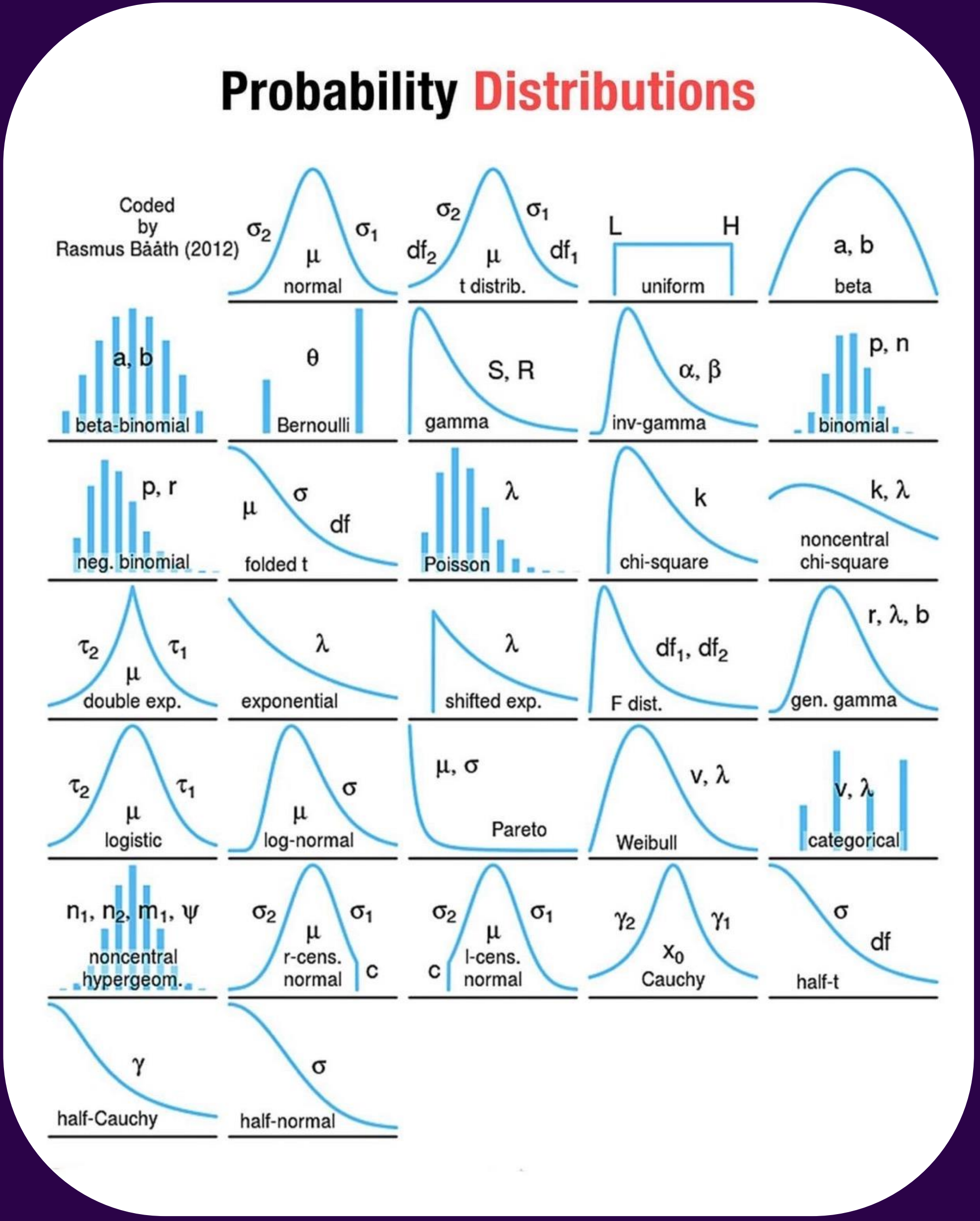
pie chart, bar chart…

Organize and show

MY FOCUS

MOTIVATION

I chose to classify the distributions in the graph since I didn't find any similar projects.



[SandhyaKrishnan02]

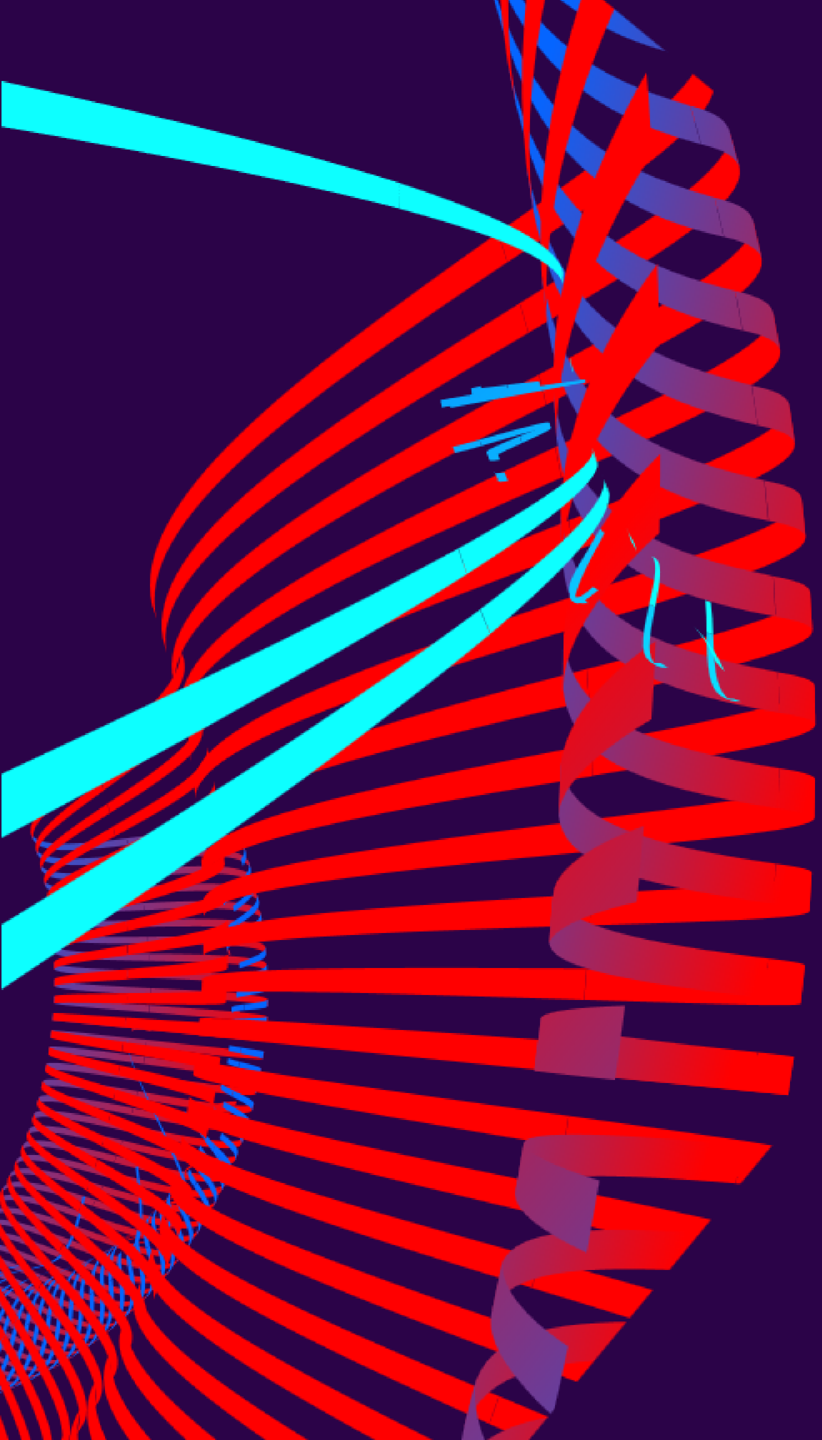
LITERATURE REVIEW

Review of neural networks and image classification

While there are many image classification research, I didn't find anything regarding the classification of graphs with probability distributions.

CS231n. (2023). (Stanford University) Retrieved January 2024, from Convolutional Neural Networks for Visual Recognition: https://cs231n.github.io/convolutional-networks/

O'Shea, K., & Nash, R. (2015). An Introduction to Convolutional Neural Networks. Aberystwyth University. Retrieved from https://arxiv.org/abs/1511.08458



OBJECTIVE

OBJECTIVE

Goals of the Project

MY FOCUS

Identify

Is it a graph?

Yes

Classify Graph

First

Identify a graph from other images.

Second

Classify the graph.

Distributions

Normal

Log-Normal

Exponential

Uniform

OBJECTIVE

Goals of the Project

Workflow of the project…

Get Data

Get 1000s of graphs of different types and distributions

Get 1000s of natural images for graph classifier

Graph Classifier

Create first model to identify whether an image is a graph or not

Distribution Classifier

Create second model to classify graphs based on their probability distributions

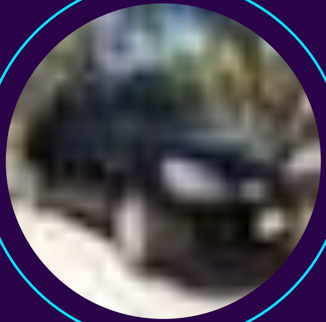
Model Evaluation

Optimize models and check validation accuracy

OBJECTIVE
Hurdles to Overcome

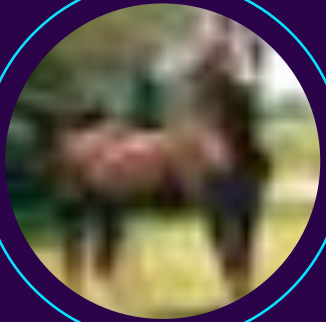
Natural Images Data

CIFAR-10

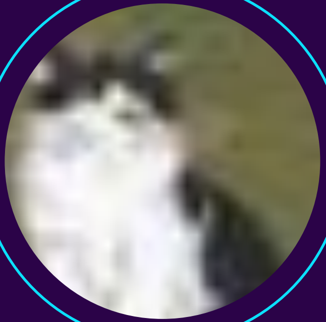


CAR

AIRPLANE



HORSE



CAT

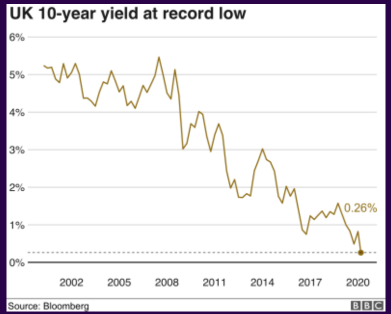
Getting natural images is easy because of quantity and quality of sources.

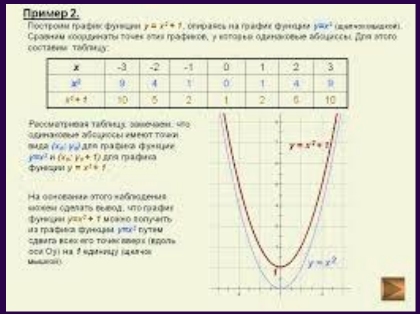
[Krizhevsky]

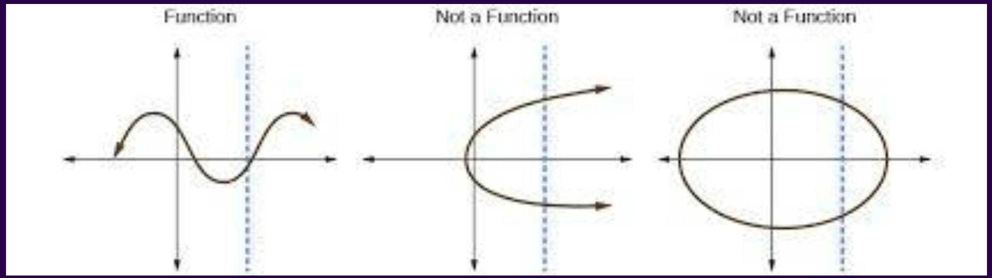
OBJECTIVE

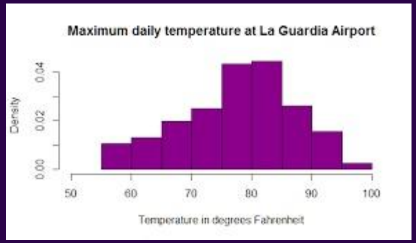
Hurdles to Overcome

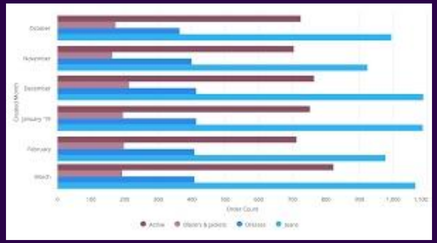
Where can I get the graph images? kaggle.com

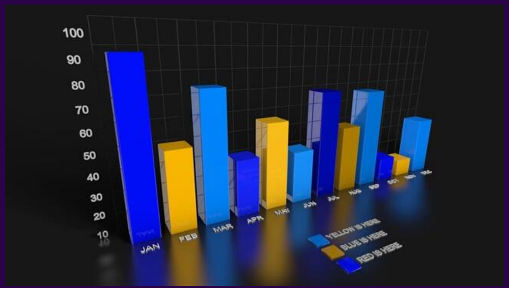


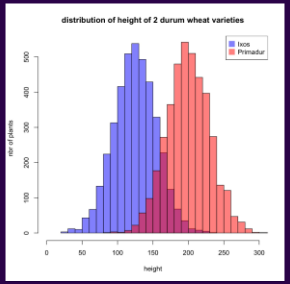


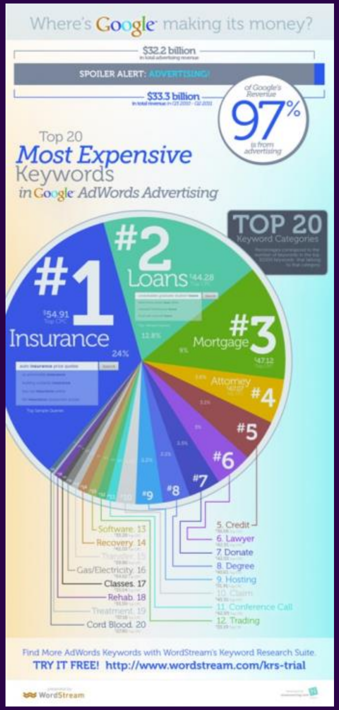












[SunEdition]

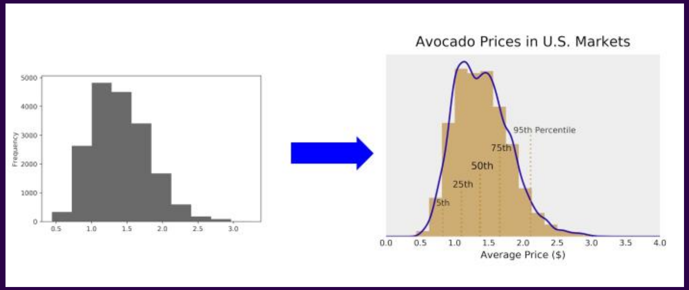
NN GRAPH CLASSIFIER

JAN 23, 2024

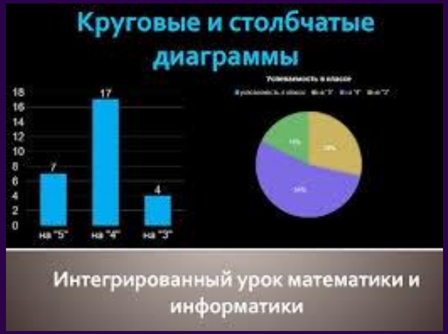
OBJECTIVE

Hurdles to Overcome

How to determine the type of a graph?



Bar chart, line chart, histogram, and log-normal distribution

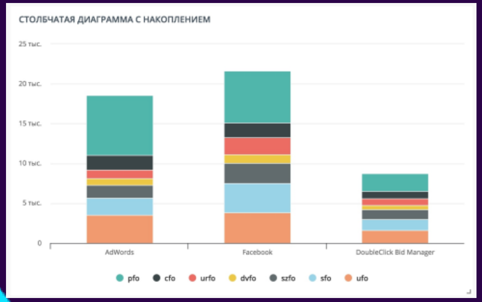


Two charts in one image

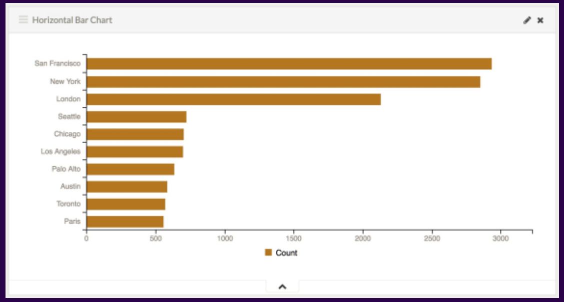
“Graph-like” images

Bar Chart Types

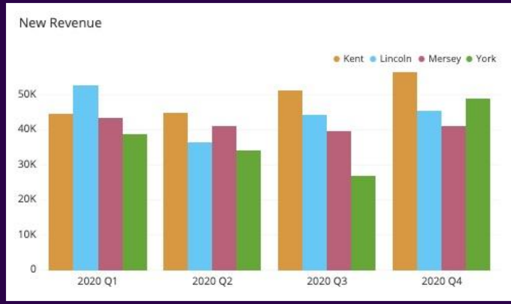
Stacked bar chart



Horizontal bar chart



Vertical bar chart



[SunEdition]

OBJECTIVE

Hurdles to Overcome

The Biggest Hurdle…

Could not find suitable dataset with probability distribution graphs.

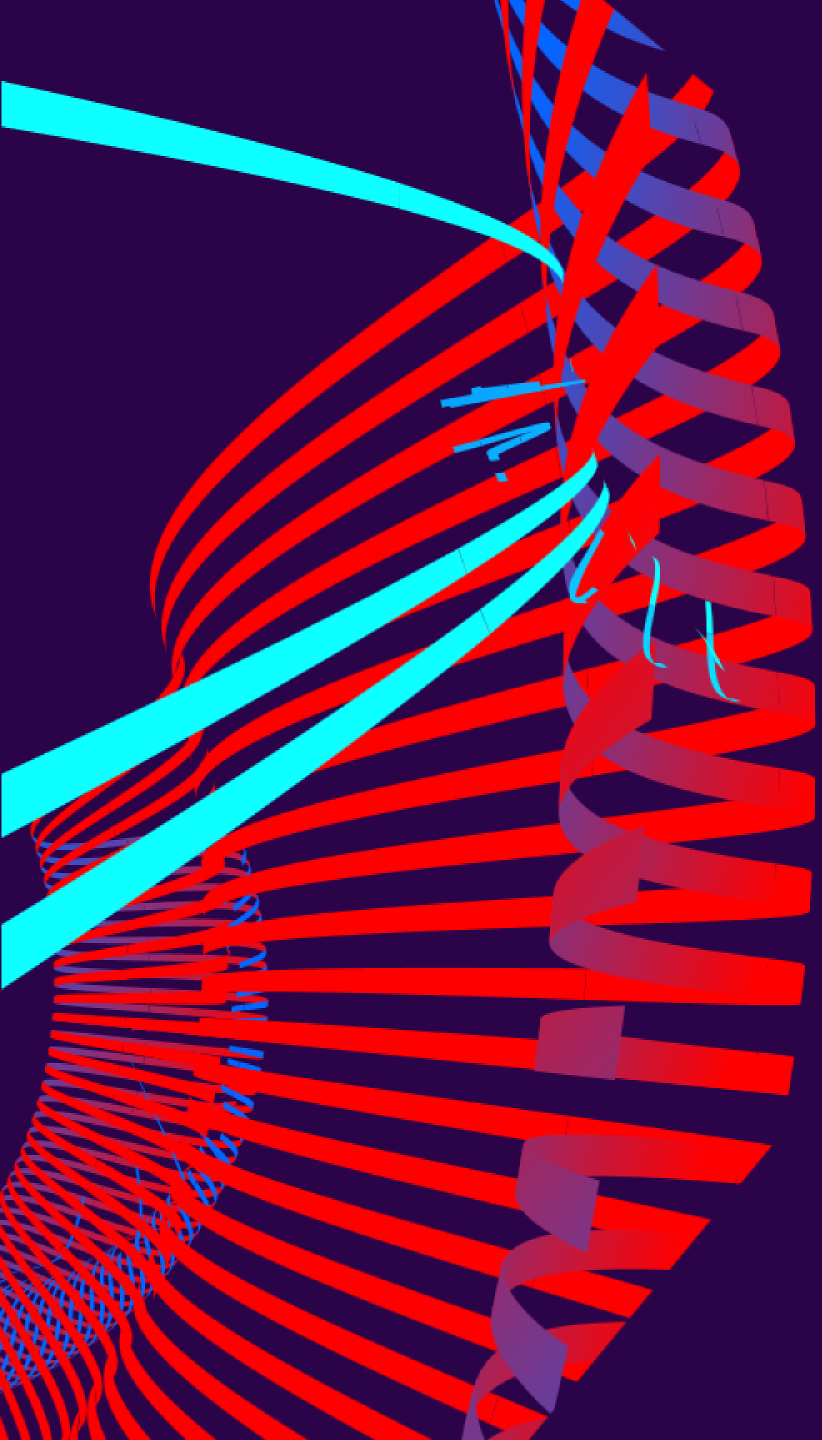
 # Solution One - Scraped

Scrape the graphs from a website, but I would run into the issue of where to scrape and how to label them time efficiently.

 # Solution Two - Generated

Use a graphing library to generate randomized graphs.

This is the most time effective solution that will lead to hopefully similar results.



DATA

DATA

3 Data Sources

CIFAR-10

Natural images of different types of objects, animals, and people

Scraped Graphs

Graph dataset scraped from various sources which weren’t stated.

Generated Graphs

Graphs with different probability distributions.

Created using a language library.

DATA

CIFAR-10 Images

CIFAR-10 Overview (CIFAR)

Canadian Institute For Advanced Research [Krizhevsky]

Count – 60,000

Dimensions – 32x32

Format – flat NumPy array

Shape – (3072,)

10 Classes – airplane, automobile, bird,

cat, deer, dog, frog, horse, ship, and truck

Updated

Count – 10,000 (first batch)

Dimensions – 32x32

Format – JPG

Shape – (32, 32, 3)

10 Classes – airplane, automobile, bird,

cat, deer, dog, frog, horse, ship, and truck

DATA

Scraped Graphs

Scraped Graphs Original

Scraped Overview (SCP)

[SunEdition]

Count – 15,786

Dimensions – various

Format - JPG

8 Classes – just image, bar chart, diagram, flow chart, graph, growth chart, pie chart, table

Updated

Count – 7,753

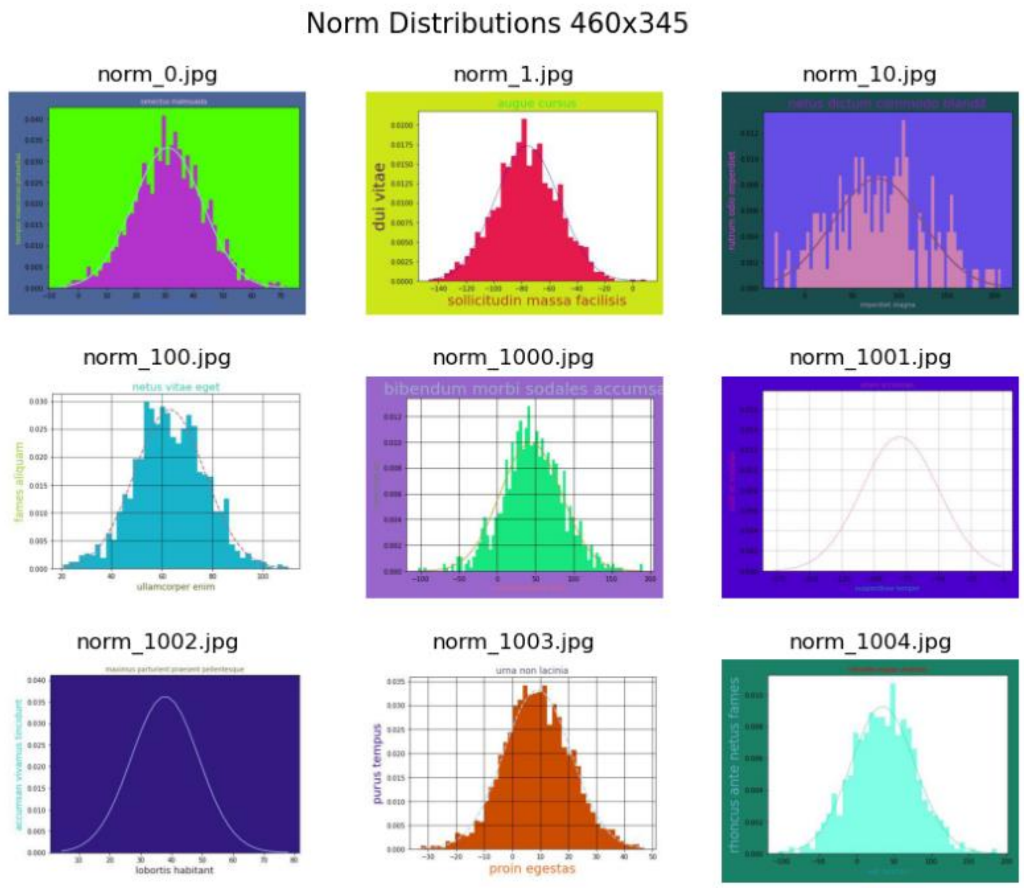
Dimensions – 32x32

Format - JPG

4 Classes – bar, diagram, graph, pie

DATA

Generated Graphs



Generated Overview (GEN)

My own dataset

Count – 8,000

Dimensions – 460x345

Format - JPG

4 Classes – norm, lognorm, exp, unif

norm: normal distribution

lognorm: log-normal distribution

exp: exponential distribution

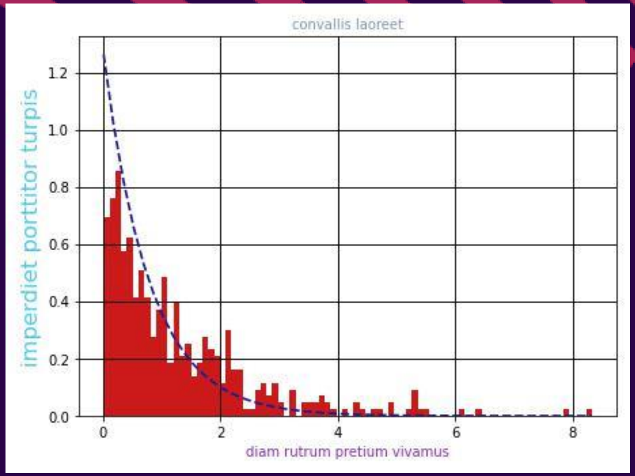
unif: uniform distribution

Updated

Dimensions – 32x32, 115x86, 153x115

METHODS

Generated Graphs



exp_199.jpg

Randomized Design

Color – RGB (0-1 for matplotlib)

Fig and Face – RGB but biased to white

Histogram – Yes/No

Line – Yes/No

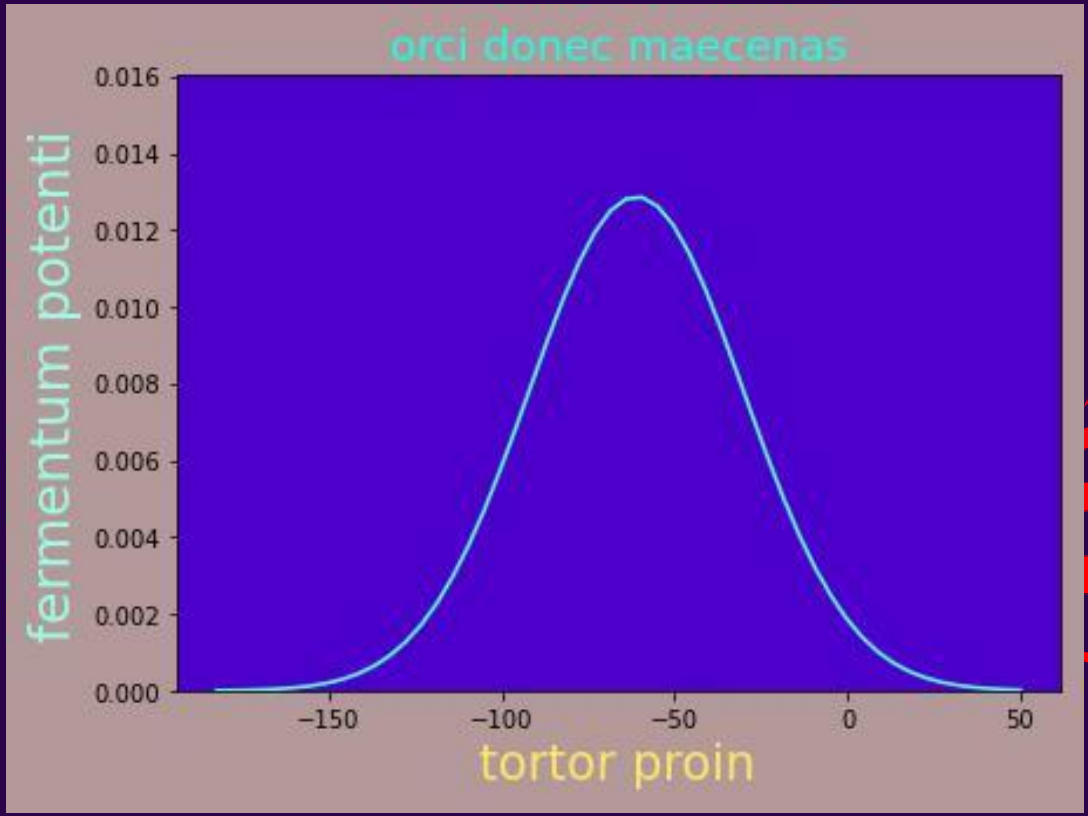
Line Style – Solid/Dash

Text – random lorum ipsum

Text Size – random

Density function parameters – random

Generated Graph Design

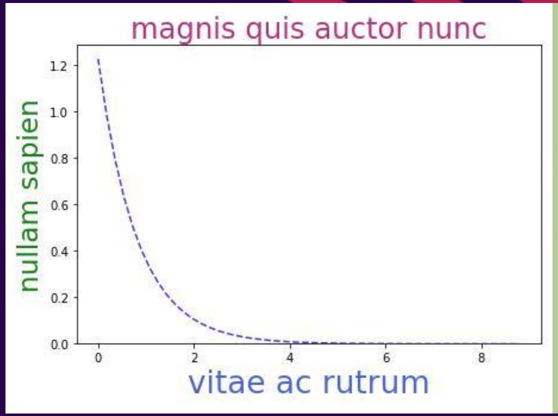


norm_796.jpg

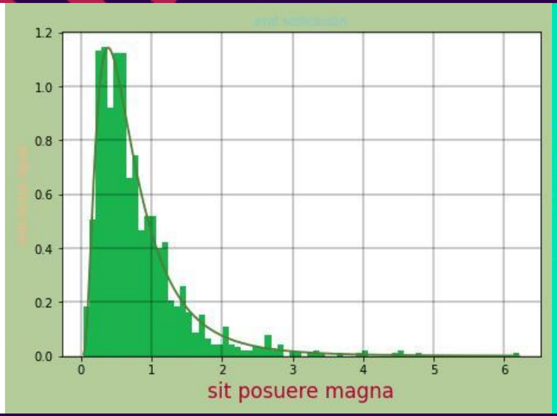
https://github.com/p-spohr/NN-Graph-Classifier/tree/main/graph_generators

NICE VS DUD GRAPHS

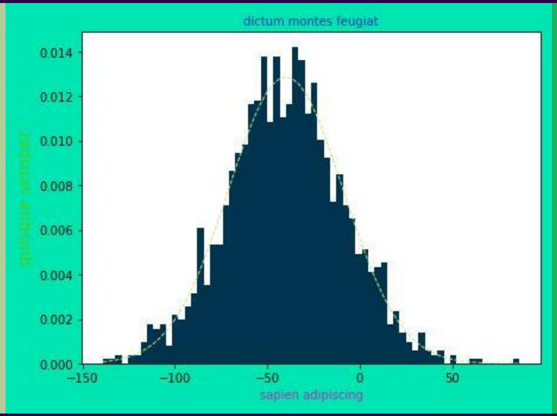
Generated Graphs



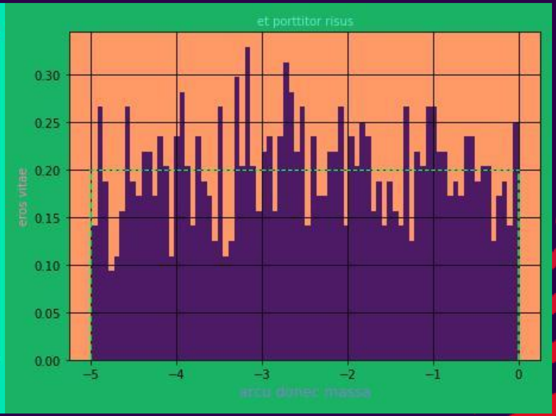
exp_146.jpg



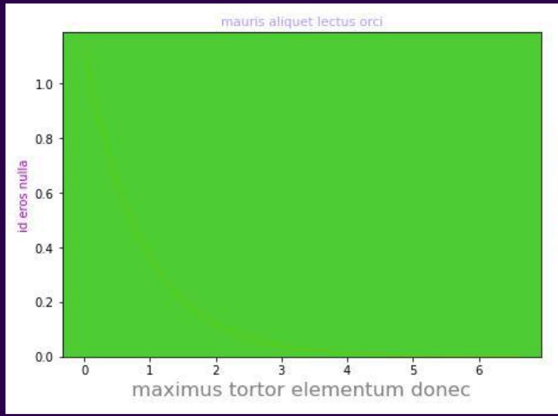
lognorm_287.jpg



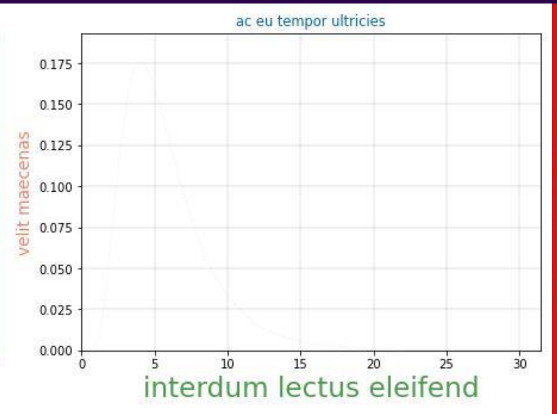
norm_226.jpg



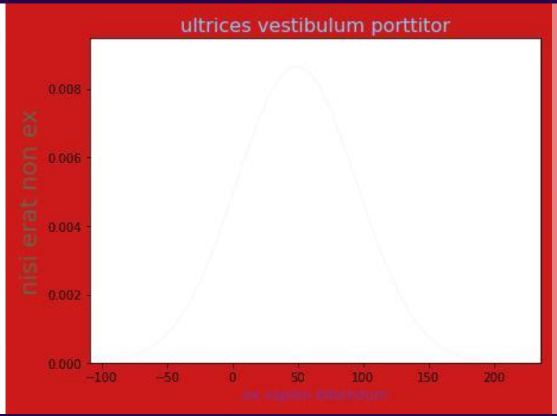
unif_1144.jpg



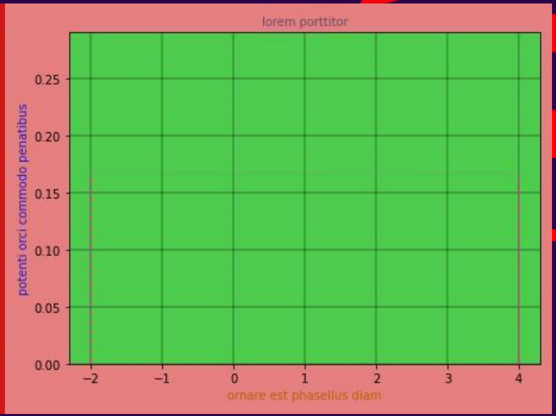
exp_31.jpg



lognorm_759.jpg



norm_1506.jpg



unif_8.jpg

METHODS

Generated Graphs

Density Functions

Normal Distribution

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

$\mu \in \mathbb{R}$

$\sigma \in \mathbb{R}_+$

Log-Normal Distribution

$$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\log(x)-\mu}{\sigma}\right)^2}$$

$\mu \in \mathbb{R}$

$\sigma \in \mathbb{R}_+$

Exponential Distribution

$$f(x) = \lambda e^{-\lambda x}$$

$\lambda \in \mathbb{R}_+$

Uniform Distribution

$$f(x) = \frac{1}{b-a}$$

$a, b \in \mathbb{R}$

$a < b$

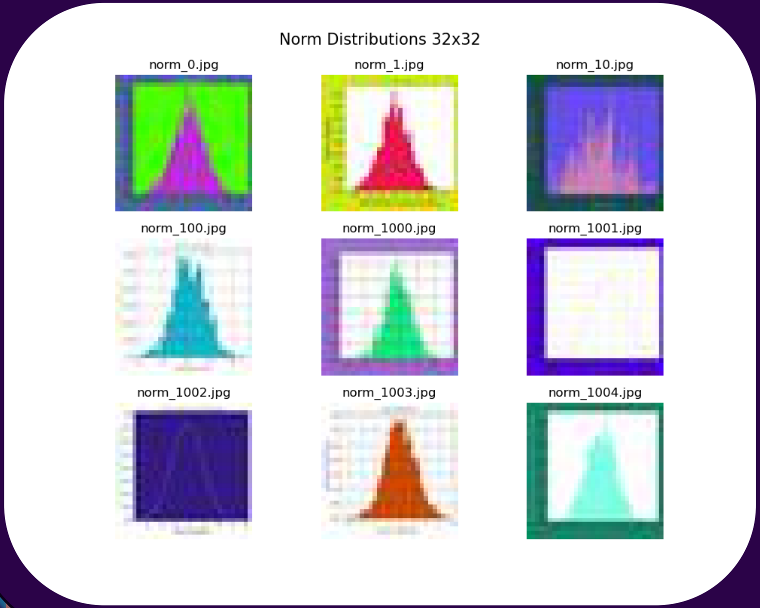
METHODS

Generated Graphs

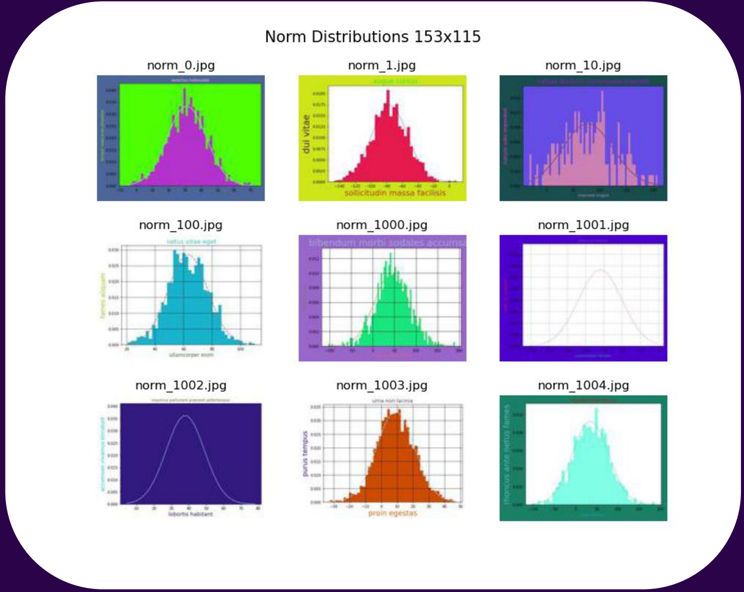
Generated Graphs Overview

Different Image Dimensions

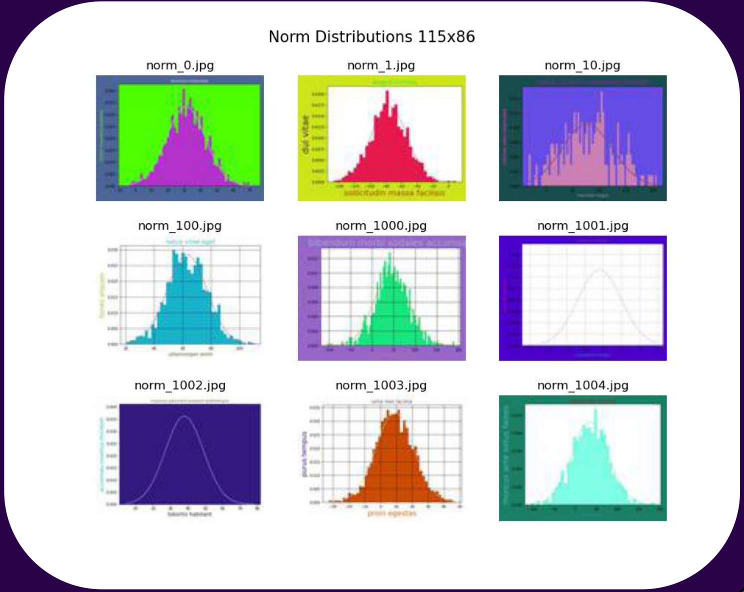
32x32



153x115



115x186

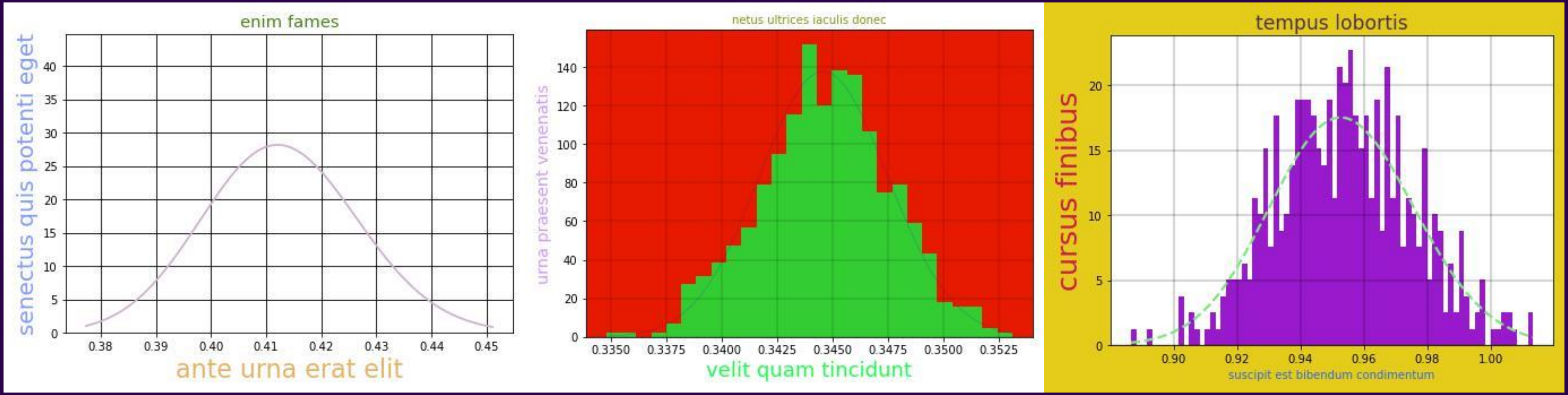


METHODS

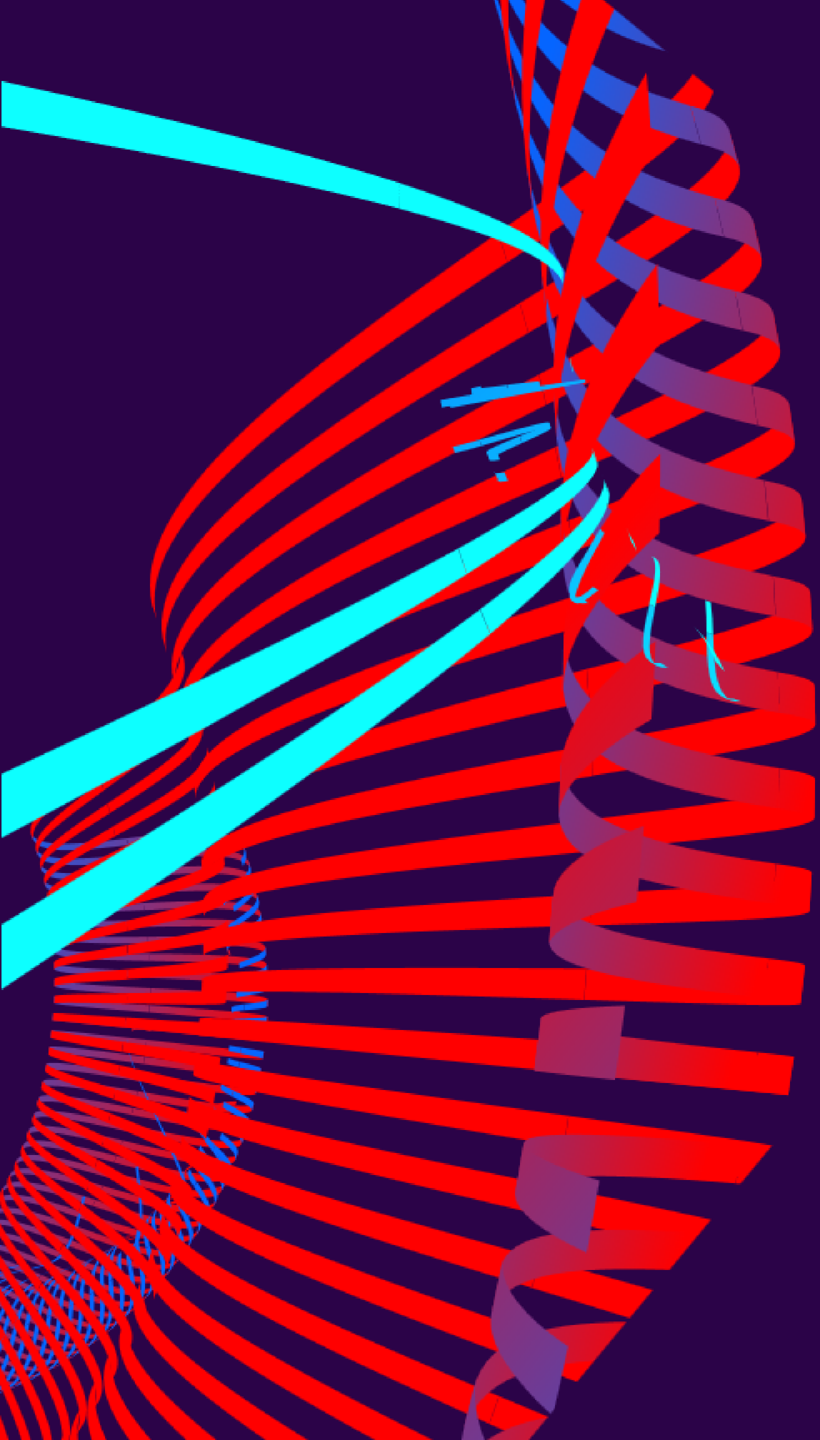
Generated Graphs

Pre-identifying possible misclassifications

Log-Normal vs. Normal Graphs



Which graph has the log-normal distribution?



METHODS

METHODS

Overview of how the goals were accomplished

- Reviewed literature available on ANNs and CNNs
- Familiarized myself with TensorFlow, Keras, and Pillow (Python library for images)
- Found natural images (CIFAR) and scraped images (SCP)
- Generated graph dataset (GEN) in 32x32, 115x86, 153x115
- Trained Simple Graph Classifiers using CIFAR, SCP, GEN datasets
- Evaluated models using untrained images
- Trained Distribution Graph Classifiers using 32x32, 115x86, 153x115
- Evaluated models to see where they misclassified
- Tested untrained images with 153x115

METHODS

Feed-Forward Neural Networks

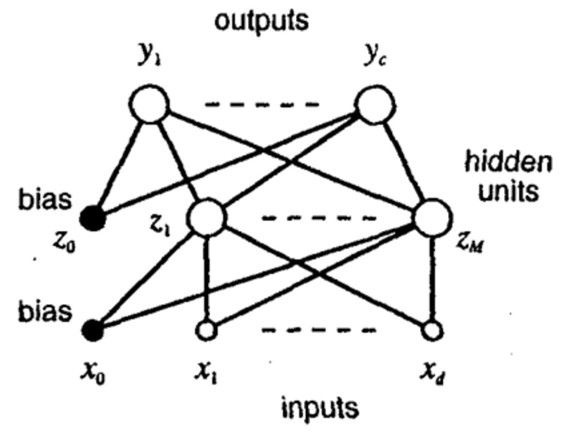


Figure 4.1. An example of a feed-forward network having two layers of adaptive weights. The bias parameters in the first layer are shown as weights from an extra input having a fixed value of $x_0 = 1$. Similarly, the bias parameters in the second layer are shown as weights from an extra hidden unit, with activation again fixed at $z_0 = 1$.

„We shall view feed-forward neural networks as providing a general framework for representing non-linear functional mappings between a set of input variables and a set of output variables."

[Bishop, 117]

Three main parts for a neural network:

Input layer $x_{1,\ldots} x_d$

Hidden layers $z_{1,\ldots} z_M$

Output layer $y_{1,\ldots} y_c$

METHODS

Feed-Forward ANN [Bishop, 116-119]

Output of the hidden-unit

$$a_j = \sum_{i=1}^{d} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$ d: Inputs

$w_{ji}^{(1)}$ weight in first layer
$w_{j0}^{(1)}$ bias for hidden unit j

$g(\cdot)$ activation function

$$z_j = g(a_j)$$

Output of the network

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_i + w_{k0}^{(2)}$$ M: Hidden-Units

Complete function for figure 4.1

$\tilde{g}(\cdot)$ activation function for the output units

$$a_k = \tilde{g}\left(\sum_{j=1}^{M} w_{kj}^{(2)} g\left(\sum_{i=1}^{d} w_{ji}^{(1)} x_i\right)\right)$$

kth output unit $y_k = \tilde{g}(a_k)$

METHODS

CNN, TensorFlow, and Python

ANN vs. CNN

The biggest difference between CNNs (convolutional neural networks) and ANNs (artificial neural networks) is that they are mostly used in image classification.

CNNs enable encoded image-specific features into the architecture (location + color), making the network more suited for image-focused tasks while also reducing the parameters required to set up the model.

[O'Shea et al.]

CNN and ANN Similarities

- ✓ Feed-Forward ANN architecture
- ✓ Dot product used for input and weights
- ✓ Back-Propagation functions the same

CNN and ANN Differences

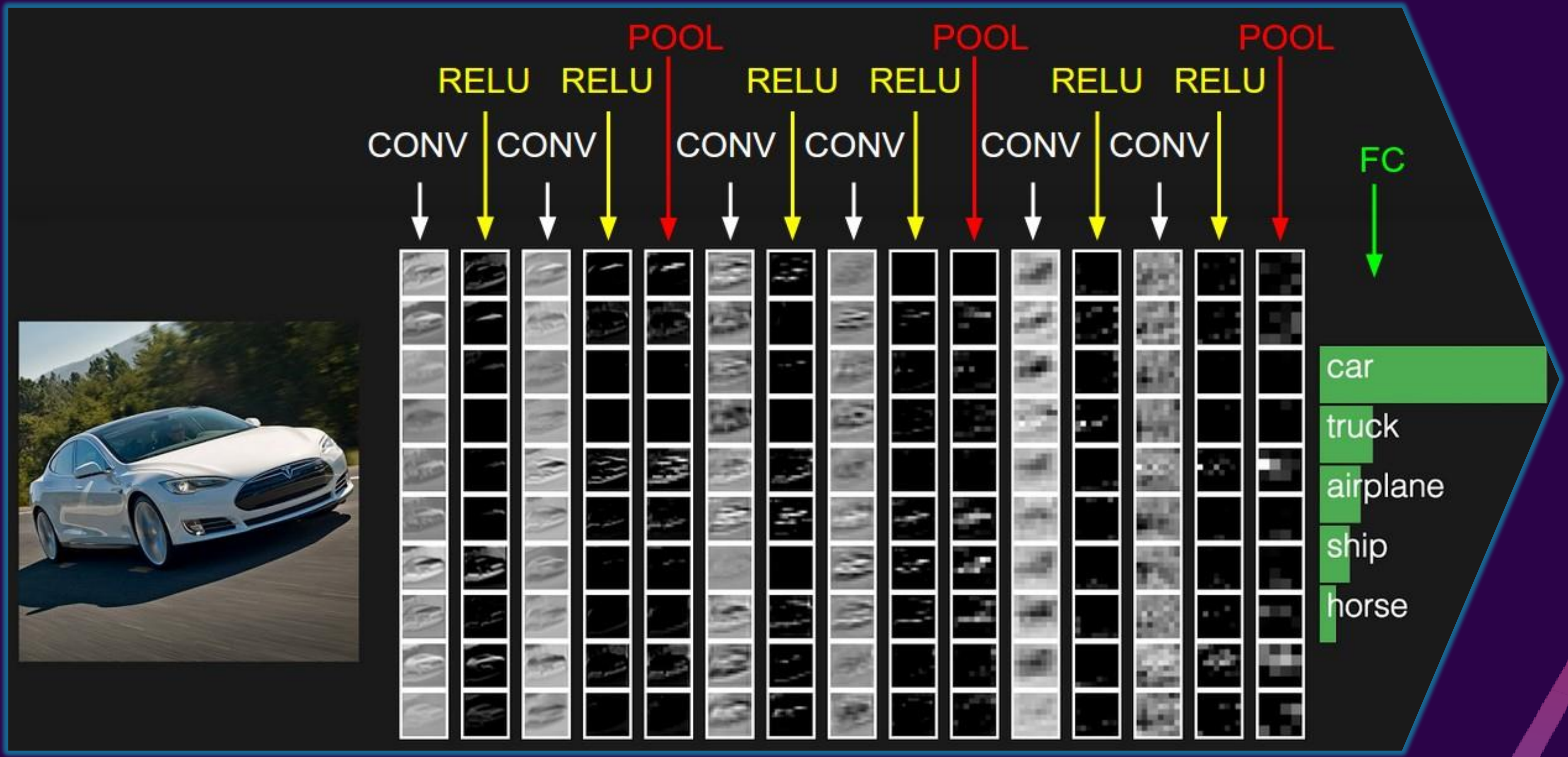
- ✓ Shape of input is based on images (H, W, RGB)
- ✓ Convolutional and pooling layers used
- ✓ ReLu activation is often used
- ✓ Neurons within a layer only connect to a small region of the layer preceding it.

METHODS

CNN, TensorFlow, and Python

Convolutional Neural Network

Layer by layer overview of using CNN to classify a car



[CS231n]

METHODS

CNN, TensorFlow, and Python

CNN Summary

Layers of CNNs:

1. Input layer – take pixel values of image (HEIGHT, WIDTH, RGB)
2. Convolutional layer – determines the output of neurons of which are connected to local regions of the input through the calculation of the scalar product between their weights and the region connected to input volume
    - *Rectified linear unit (ReLu)* – aims to apply an 'elementwise' activation function such as sigmoid to the output of the activation produced by the previous layer
3. Pooling layer – downsampling along the spatial dimensionality of the given input, reducing number of parameters within that activation
4. Fully-connected layers (output layer) – performs the same duties found in standard ANNs and attempts to produce class scores from the activations to be used for classification

[O'Shea et al.]

METHODS

CNN, TensorFlow, and Python

How to Understand CNN?

…first understand the input.

Images have a shape of (X, Y, RGB) where RGB determines the color.
Each color consist of a combination of red, green and blue.

(255, 255, 255)

(255, 0, 0)

(0, 255, 0)

(0, 0, 255)

(0, 0, 0)

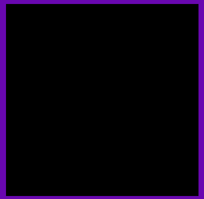
METHODS

CNN, TensorFlow, and Python

Dimensions and RGB

Now we can look at the shape.

We are working with tensors instead of matrices now.

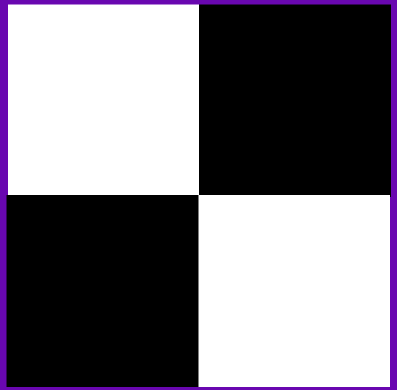


((0, 0, 0))

(1,1,3)

((255, 255, 255), (0, 0, 0))

(1,2,3)



(((255, 255, 255), (0, 0, 0)),
((0, 0, 0), (255, 255, 255)))

(2,2,3)

METHODS

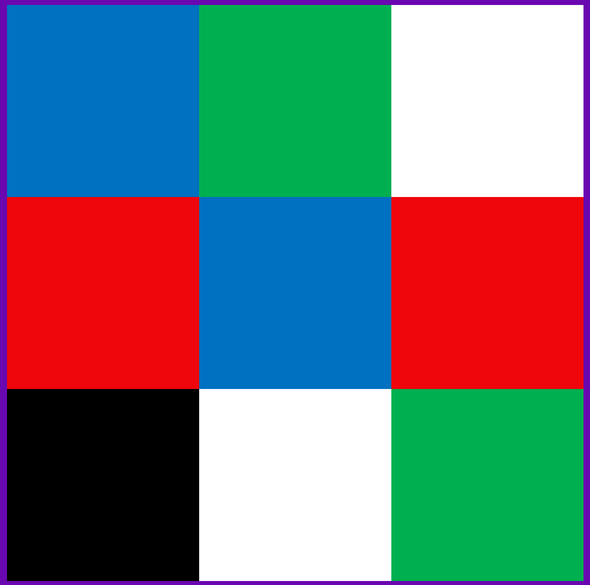
CNN, TensorFlow, and Python

Image Quiz

Can you correctly write out this image as a tensor?

Red	(255, 0, 0)
Green	(0, 255, 0)
Blue	(0, 0, 255)
White	(255, 255, 255)
Black	(0, 0, 0)

Also, what is its shape?



Answer

METHODS

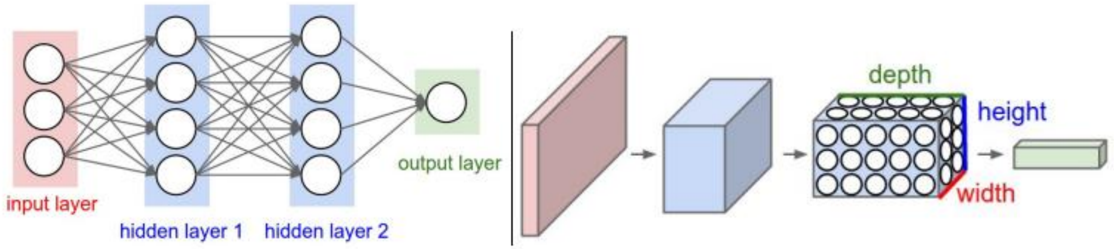
CNN, TensorFlow, and Python

Convolutional Layer

The original image gets transformed layer by layer from the original pixel values to the final class scores.

By the output layer the full image will be reduced into a single vector of class scores, arranged along the depth dimension.

Example:
The final output layer for CIFAR-10 would have dimensions of 1x1x10.



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

[CS231n]

Most Important Parts:
- **Filter – number of output filters**
- **Kernel – size of the filter**
- **Stride – number of pixels the kernel slides**
- **Padding – how zeros are added**

METHODS

CNN, TensorFlow, and Python

Convolutional Layer

$$Output\ Volume\ Size = \frac{(W - F + 2P)}{S} + 1$$

W := input volume size
F := receptive field size (kernel)
P := amount of zero padding
S := stride of filter
K := filter count

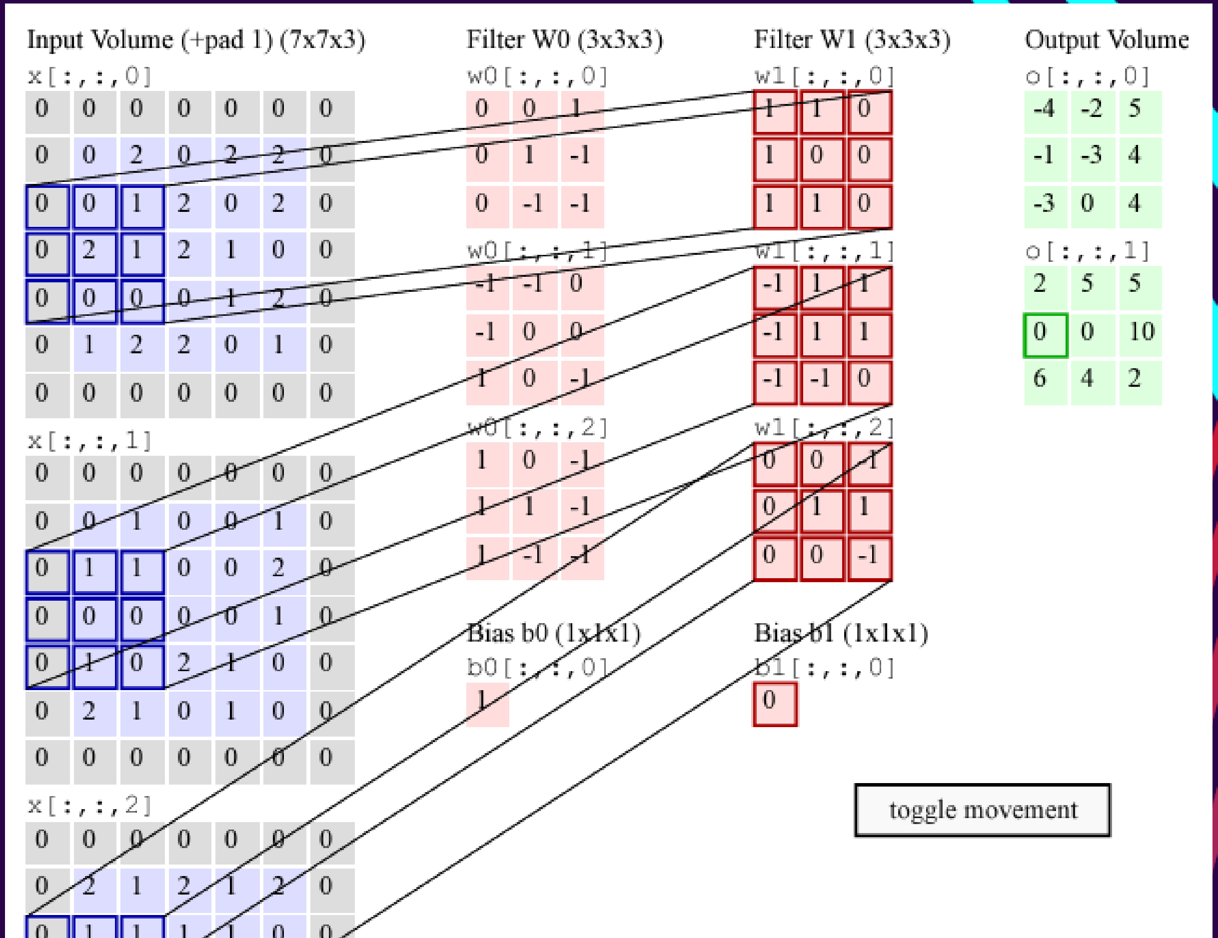
Example:
Width = 5
Height = 5
RGB = 3
K = 2
F = 3
S = 2
P = 1

$$OVS = \frac{(5 - 3 + 2 \cdot 1)}{2} + 1 = 3$$

R

G

B



$$\sum Kernal \odot Filter\ Weights = OV$$

elementwise

[CS231n]

METHODS

CNN, TensorFlow, and Python

Rectified Linear Units (ReLU)

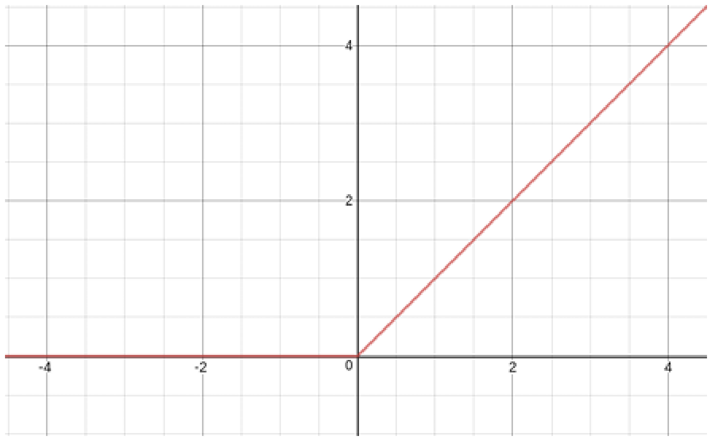


Figure 1: The Rectified Linear Unit (ReLU) activation function produces 0 as an output when $x < 0$, and then produces a linear with slope of 1 when $x > 0$.

[Agarap]

$$f(x) = x^{+} = max(0, x) = \frac{x + |x|}{2} = \begin{cases} x \; if \; x > 0, \\ 0 \; otherwise, \end{cases}$$

Why ReLu?

- eliminates complex calculations
- reduces processing demands
- model can learn in less time
- promotes sparsity

Sparsity refers to a scenario where most of the cell entries in a matrix are zero [Giskard].

METHODS

CNN, TensorFlow, and Python

Max Pooling

The feature map output of convolutional layers record the precise position of features in the input. *Small movements in the position of the feature will result in a different feature map.*

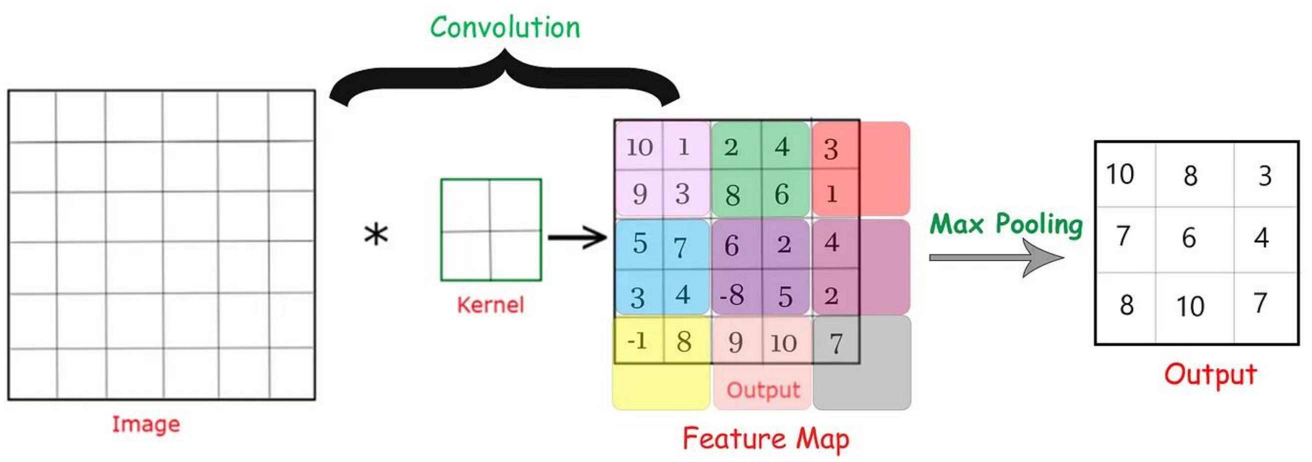
Downsampling:

- ➢ lower resolution version of an input signal is created
- ➢ contains the large or important structural elements without the fine detail that may not be as useful

[Brownlee]

Maximum pooling calculates the maximum, or largest, value in each patch of each feature map.

$$MP = max(f_i) \; for \; i = 1, \ldots, F \text{ feature map patches}$$



[Kumar, Credit: Codicals]

METHODS

CNN, TensorFlow, and Python

The **cross-entropy loss function** in classification calculates how accurate our machine learning or deep learning model is by defining the difference between the estimated probability with our desired outcome.

[The 365 Team]

Binary Cross-Entropy Loss

Only two classes

$$BCEL = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i log(p_i) + (1 - y_i) log(1 - p_i)\right)$$

Categorical Cross-Entropy Loss

More than two classes

$$CCEL = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{i,j} \cdot \log(p_{i,j})$$

C := number of classes

N := number of rows

y := indicator if class label j is the correct classification for observation i

p := predicted probability observation i is of class j

METHODS

CNN, TensorFlow, and Python

Adam Optimizer

"…an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments [Kingma et al.]."

Used to minimize the loss function.

$\alpha \coloneqq learning\ rate$

$\beta_1 \coloneqq beta\ one$

$\beta_2 \coloneqq beta\ two$

$\varepsilon \coloneqq epsilon$

Pros:

- only requires first-order gradients
- little memory requirement

minimizes the loss function during the training of neural networks

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

[Kingma et al.]

TENSORFLOW + KERAS API

TensorFlow – end-to-end platform for building ML models

Keras – deep learning API written in Python (compatible with TensorFlow)

https://www.tensorflow.org/api_docs/python/tf

https://keras.io/api/

METHODS

CNN, TensorFlow, and Python

PYTHON + LIBRARIES

NumPy – math + ndarray + random

pandas – data handling

Matplotlib – graphing

Pillow – image processing
(TensorFlow returns PIL.Image)

https://numpy.org/doc/stable/reference/index.html

https://matplotlib.org/stable/index.html

https://pandas.pydata.org/docs/reference/index.html

https://pillow.readthedocs.io/en/stable/

METHODS

CNN, TensorFlow, and Python

Library Versions

python	3.10.13
tensorflow	2.10.0
numpy	1.26.0
pandas	2.1.1
matplotlib	3.8.0
pillow	10.0.1

Other Programs Used

Visual Studio Code (code editor)

Anaconda (Python environments)

Jupyter Notebook (interactive code)

METHODS

Simple Graph Classifiers

CIFAR: natural images

Total = 10000

SCP: scraped graphs

Total = 7753

GEN: generated graphs

Total = 8000

1: Model 1

CIFAR_SCP_1	Total = 17753
x = 32x32 images	Train = 14203
y = [graph, natural]	Val = 3550

GEN_SCP_1	Total = 15753
x = 32x32 images	Train = 12603
y = [generated, scraped]	Val = 3150

CIFAR_GEN_1	Total = 18000
x = 32x32 images	Train = 14400
y = [graph, natural]	Val = 3600

CIFAR_GEN_SCP_1	Total = 25753
x = 32x32 images	Train = 20603
y = [generated, natural, scraped]	Val = 5150

https://github.com/p-spohr/NN-Graph-Classifier/tree/main/classifiers/simple_graph_classifiers

METHODS

Simple Graph Classifiers

1: Model 1

Same model for simple graph classifiers and distribution graph classifiers.

```
model = Sequential([
    # Standardize values to be in the [0, 1] range by using tf.keras.layers.Rescaling
    layers.Rescaling(1./255, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes)
])
```

"same" results in padding with zeros evenly to the left/right or up/down of the input.

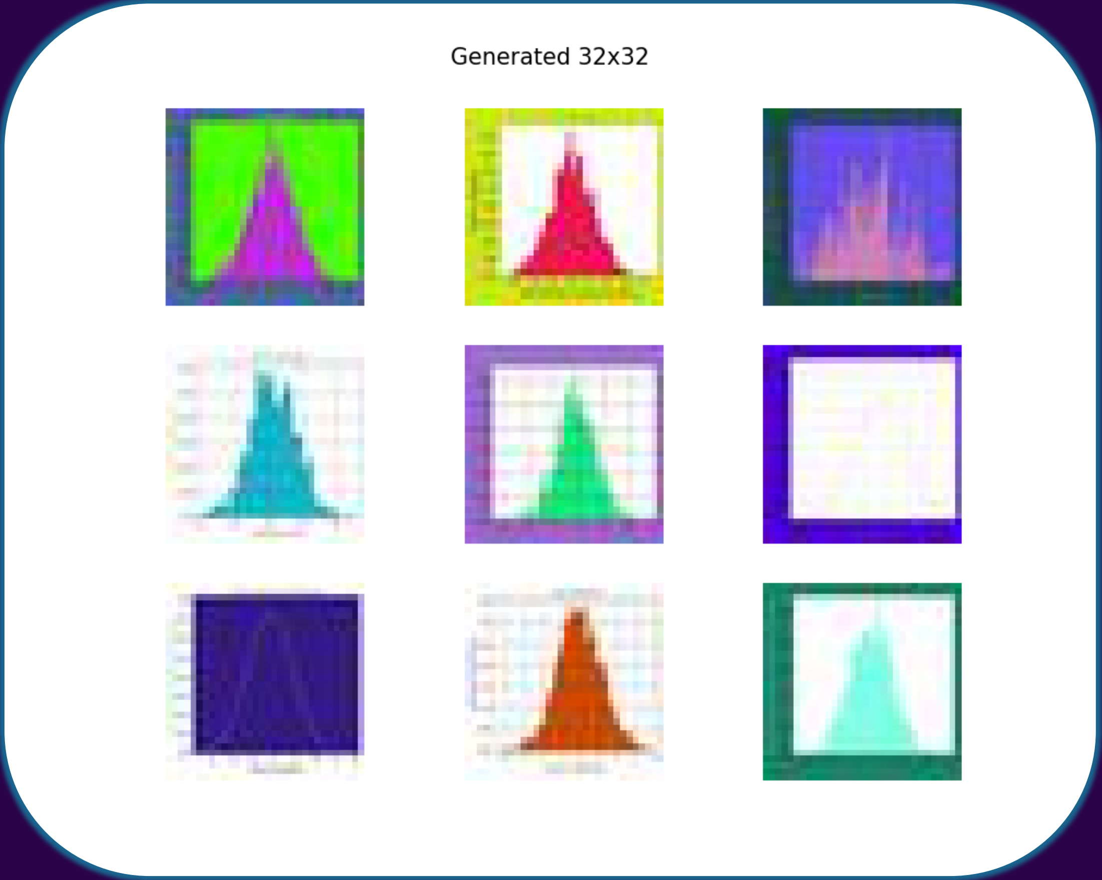
METHODS

Simple Graph Classifiers

CIFAR_GEN_1

Image examples

Input layer := x = (32, 32, 32, 3)



y = graph

y = natural

METHODS

Simple Graph Classifiers

Load image datasets

```
tf.keras.utils.image_dataset_from_directory(
    directory,
    labels='inferred’, # subfolder names
    label_mode='int',
    class_names=None,
    color_mode='rgb’, # jpg has no alpha
    batch_size=32,
    image_size=(32, 32), # (height, width)
    shuffle=True,
    seed=None,
    validation_split=None, # 0-1 for validation
    subset=‘both’, # ‘both’ returns (‘train’, ‘val’)
    interpolation='bilinear',
    follow_links=False,
    crop_to_aspect_ratio=False,
    **kwargs
)
```

CIFAR_SCP_1 & CIFAR_GEN_1

Name	Date modified	Type	Size
graph	1/5/2024 2:31 PM	File folder	
natural	1/5/2024 2:30 PM	File folder	

CIFAR_GEN_SCP_1

Name	Date modified	Type	Size
generated	1/5/2024 3:16 PM	File folder	
natural	1/5/2024 3:16 PM	File folder	
scraped	1/5/2024 3:17 PM	File folder	

https://www.tensorflow.org/api_docs/python/tf/keras/utils/image_dataset_from_directory

METHODS

Simple Graph Classifiers

Code Steps Overview

Similar steps for all models except the datasets and targets are different

1) Initialize constant parameters and directory path → 2) Get dataset from directory path

```
# dir path to folders (folder name == label)
DATASET_PATH = "C:\\Users\\pat_h\\htw_berlin_datasets\\CIFAR_GEN_1_DATASET"

# choose image dimensions
BATCH_SIZE : int = 32
IMG_HEIGHT : int = 32
IMG_WIDTH : int = 32

# choose data validation percentage, seed, and epochs
VAL_SPLIT : int = 0.2
SEED : int = 123
EPOCHS : int = 3
```

```
# get training dataset and validation dataset from directory path
train_ds, val_ds = tf.keras.utils.image_dataset_from_directory(
    DATASET_PATH,
    validation_split=VAL_SPLIT,
    labels='inferred',
    subset='both',
    seed=SEED,
    image_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE
)
```

3) Build model → 4) Compile and fit model to data

```
model = Sequential([
    # Standardize values to be in the [0, 1] range by using tf.keras.layers.Rescaling
    layers.Rescaling(1./255, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes)
])
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
# returns History object
fitted_model = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=EPOCHS
)
```

https://github.com/p-spohr/NN-Graph-Classifier/blob/main/classifiers/simple_graph_classifiers/CIFAR_GEN_1.py

CIFAR_GEN_1 SUMMARY

```
Model: "sequential"
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 32, 32, 3)         0

 conv2d (Conv2D)             (None, 32, 32, 16)        448

 max_pooling2d (MaxPooling2D  (None, 16, 16, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 16, 16, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 32)         0
 2D)

 conv2d_2 (Conv2D)           (None, 8, 8, 64)          18496

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 64)         0
 2D)

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 128)               131200

 dense_1 (Dense)             (None, 2)                 258

=================================================================
Total params: 155,042
Trainable params: 155,042
Non-trainable params: 0
_________________________________________________________________
```

METHODS

Distribution Graph Classifiers

DIST = distribution classifier *WIDTH x HEIGHT = dimensions of image 1 = Model 1

Input

Total = 8000

Train = 6400

Val = 1600

DIST_32x32_1
x = 32x32 images

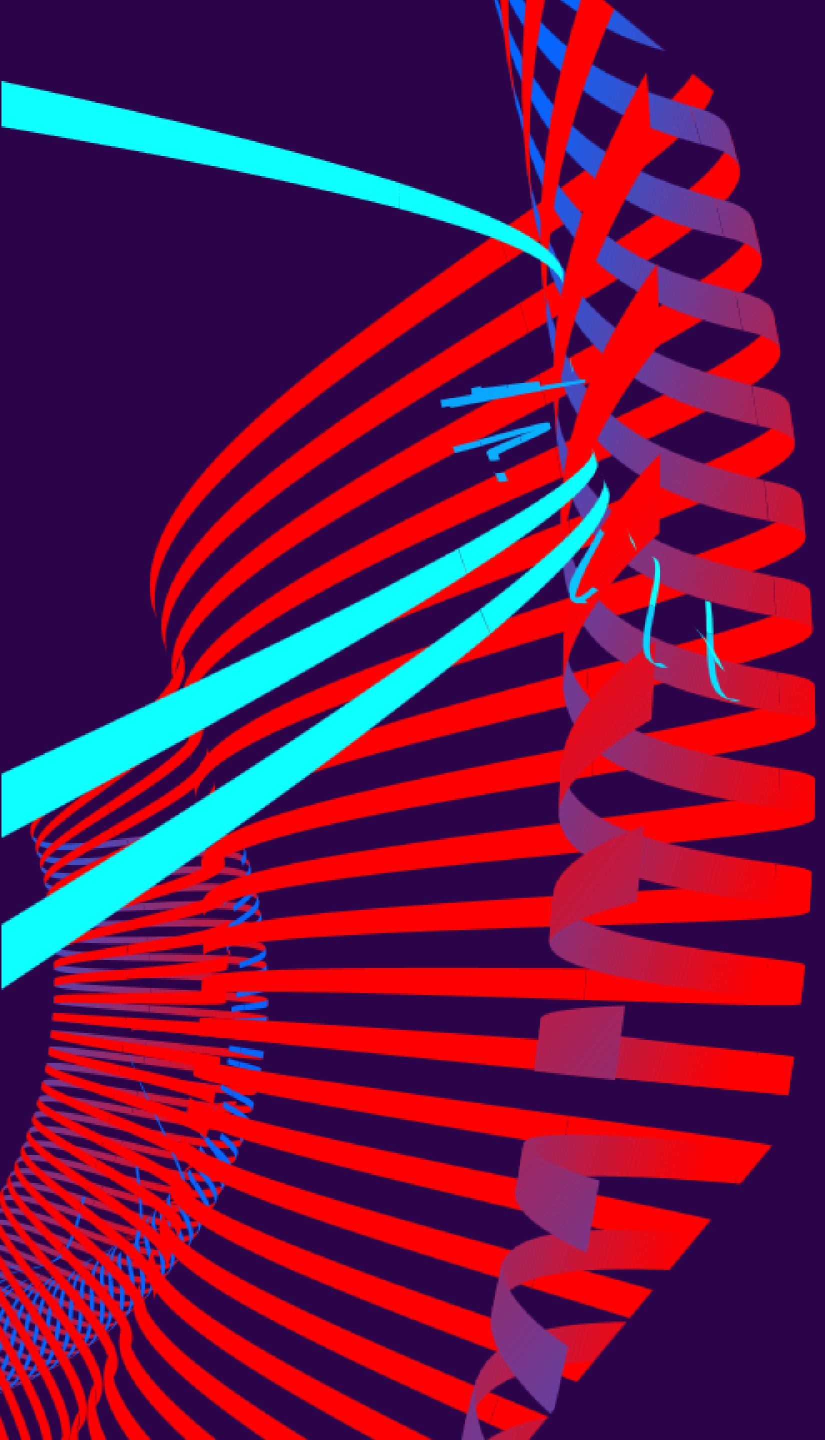
DIST_115x86_1
x = 115x86 images

DIST_153x115_1
x = 153x115 images

y = [exp, lognorm, norm, unif]

*IMPORTANT: dimensions are reversed in TensorFlow
input_shape = (HEIGHT, WIDTH)

https://github.com/p-spohr/NN-Graph-Classifier/tree/main/classifiers/distribution_graph_classifiers



RESULTS

RESULTS OVERVIEW

Simple Graph Classifiers

CIFAR_SCP_1

CIFAR_GEN_1

GEN_SCP_1

CIFAR_GEN_SCP_1

Distribution Graph Classifiers

DIST_32x32_1

DIST_115x86_1

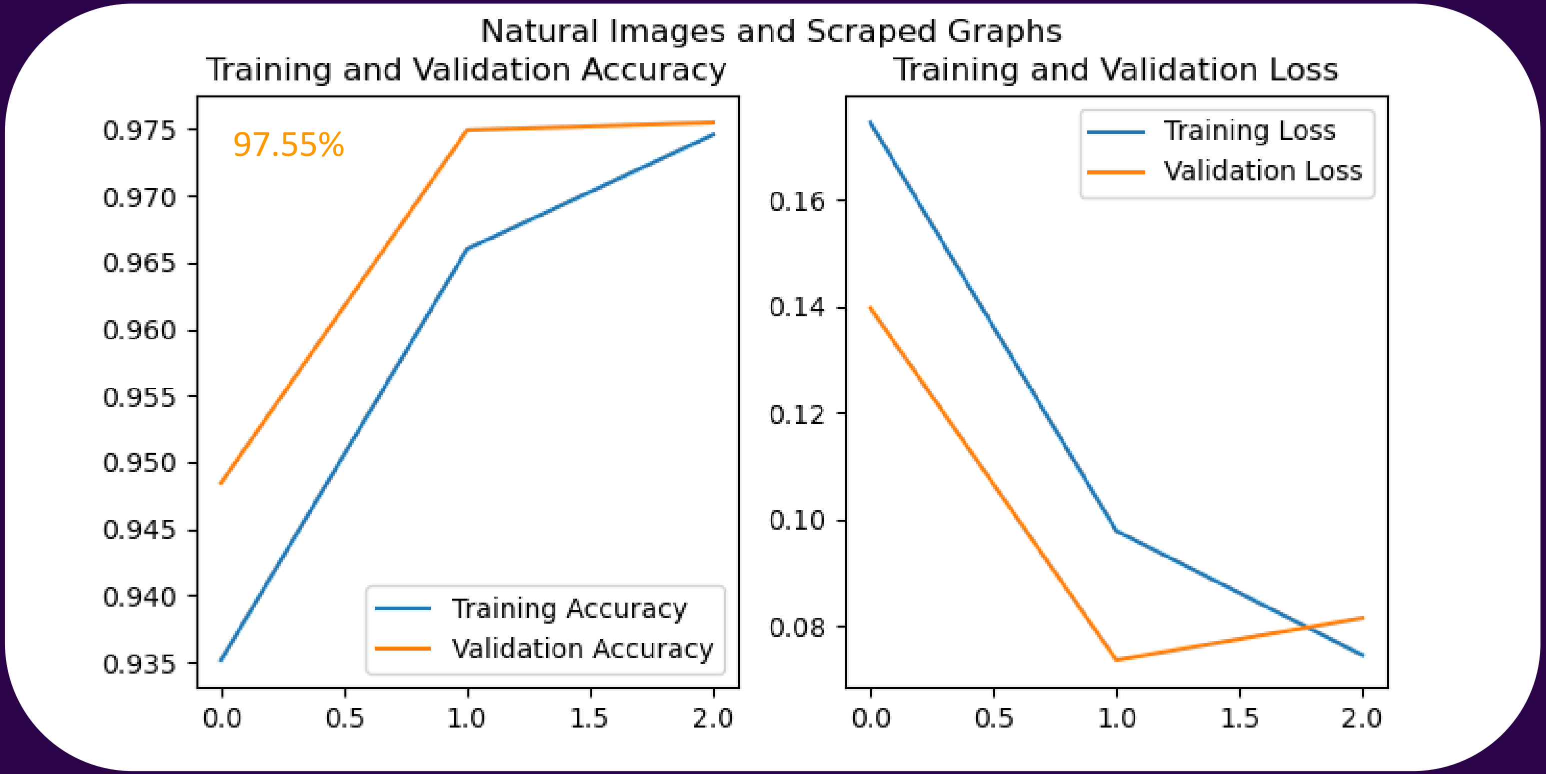
DIST_153x115_1

SIMPLE GRAPH CLASSIFIERS

RESULTS

CIFAR_SCP_1

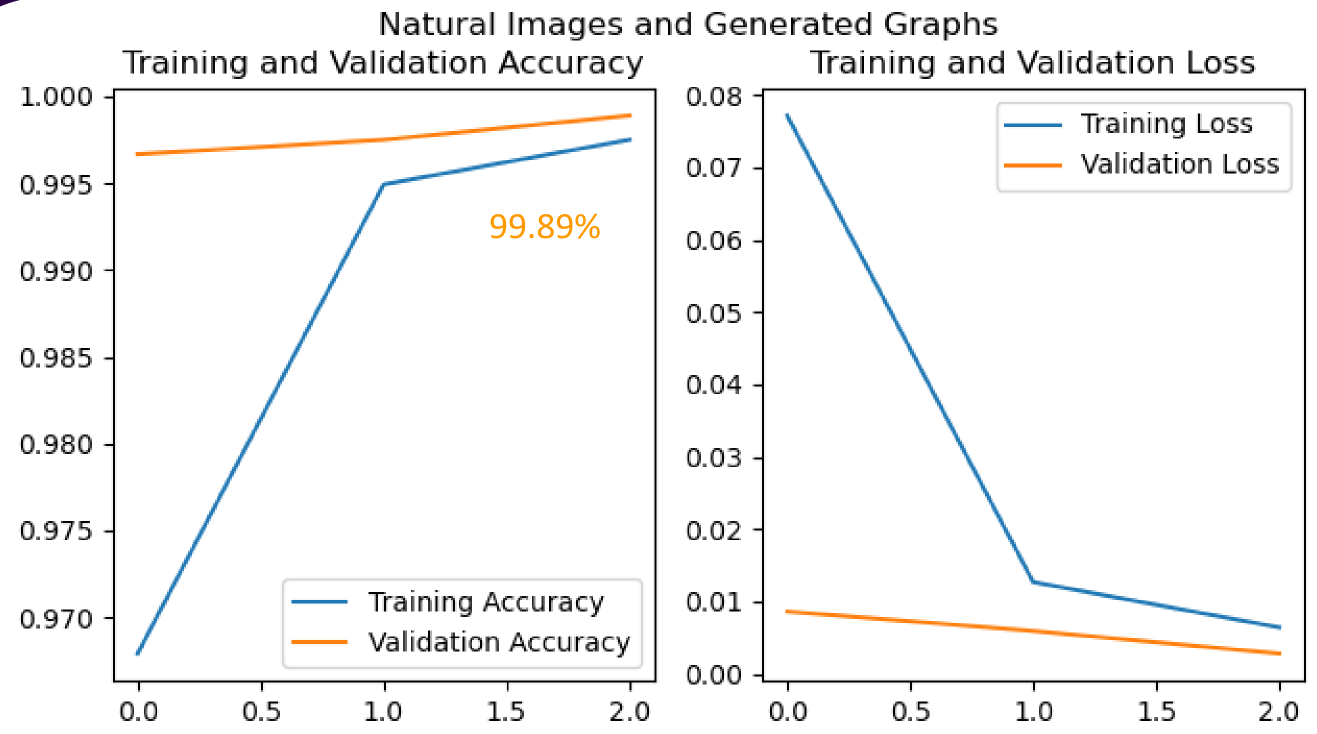
Natural Images and Scraped Graphs



High validation accuracy after 3 epochs

CIFAR_GEN_1

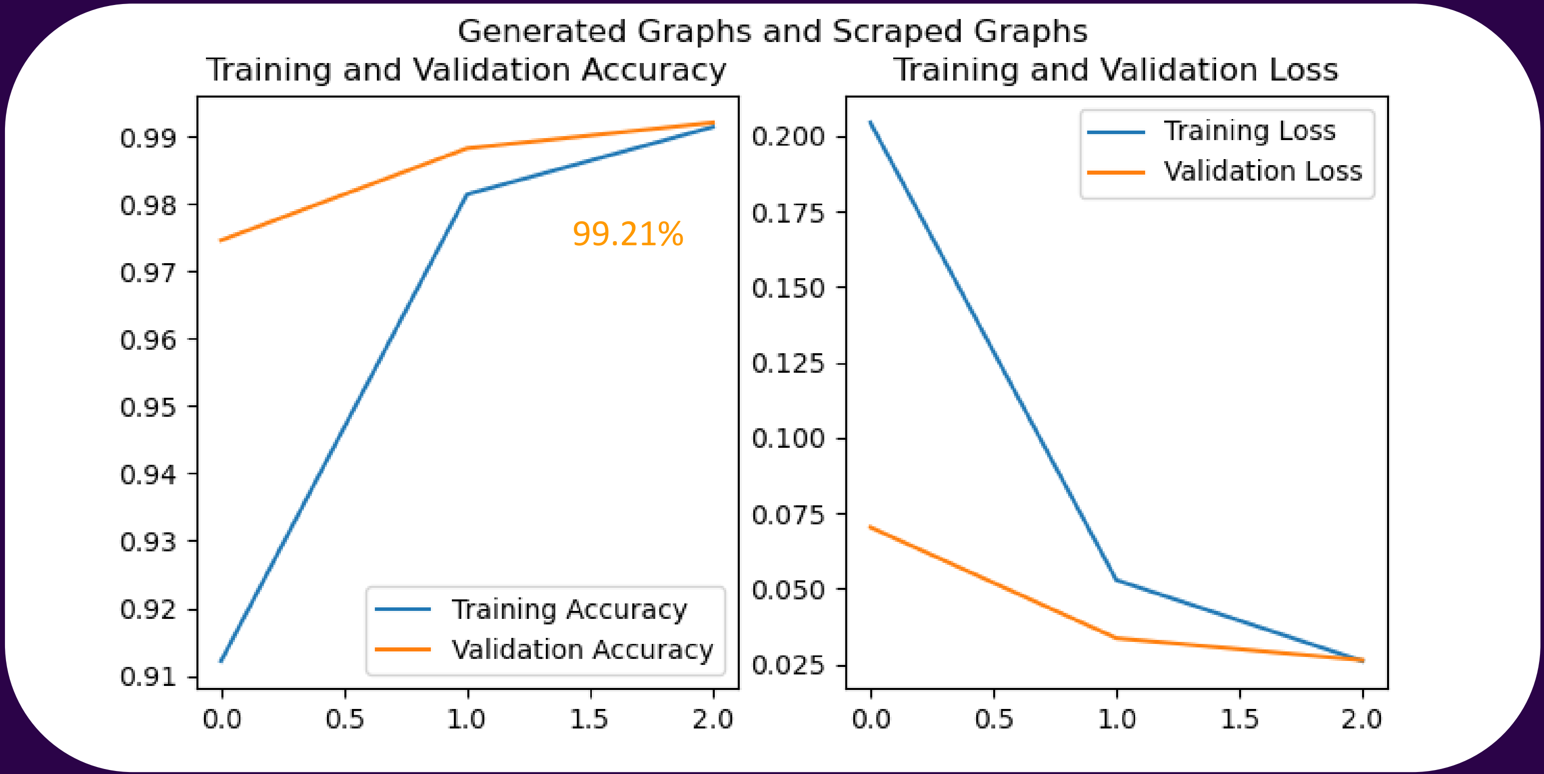
Natural Images and Generated Graphs (my graphs)



Even my generated graphs can be classified accurately

GEN_SCP_1

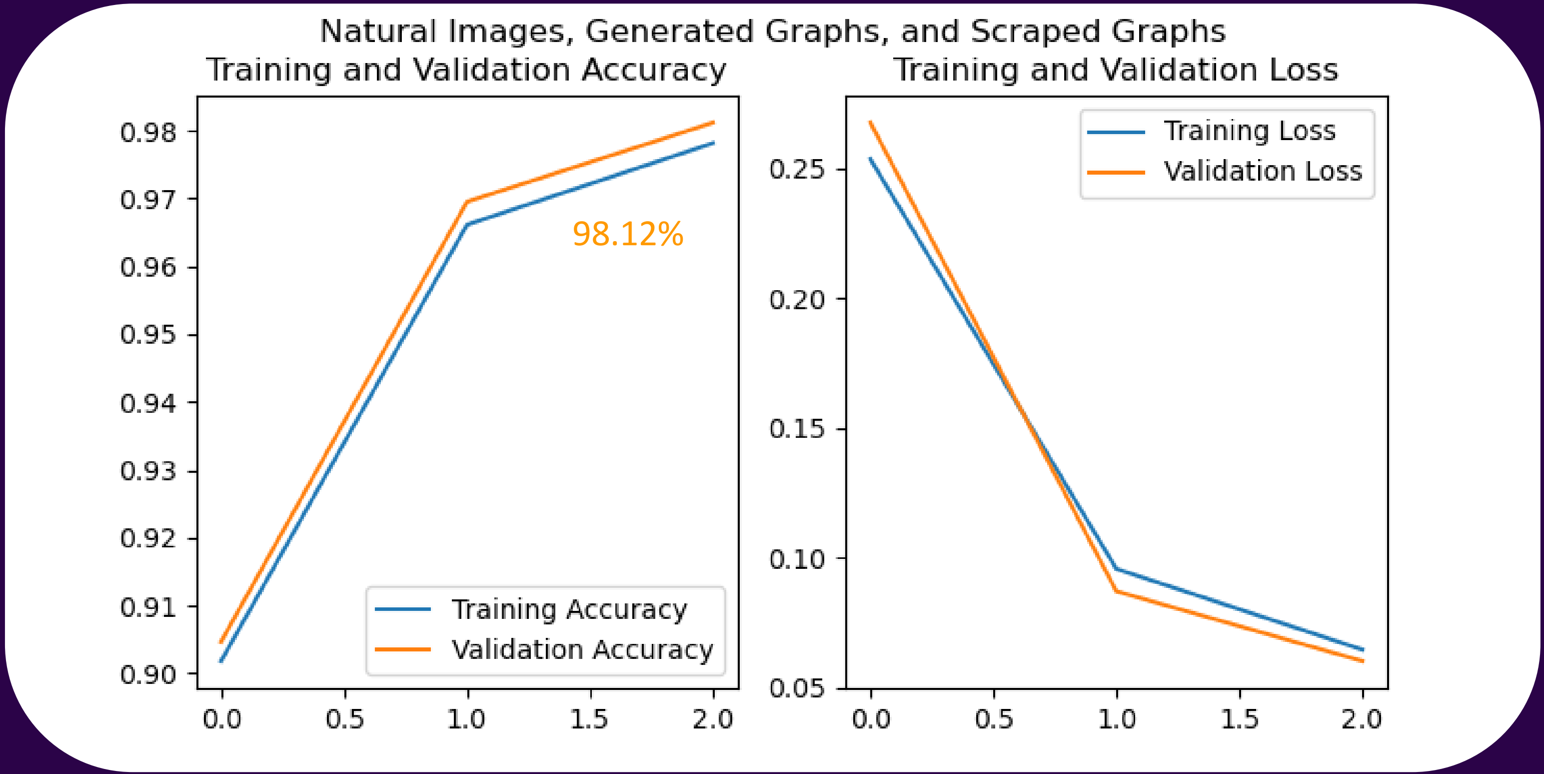
Generated Graphs and Scraped Graphs



This was surprising! The generated graphs are quite distinguishable from the scraped graphs

CIFAR_GEN_SCP_1

Natural Images, Generated Graphs, and Scraped Graphs



For good measure –a model to classify all three

EVALUATIONS

Can the models predict if an image from the other graph dataset is a graph or not?

Generated Graphs in CIFAR_SCP_1

Prediction	Count
True (graph)	6475
False (natural)	1525

Accuracy: 80.9375%

Best case is 100% accuracy

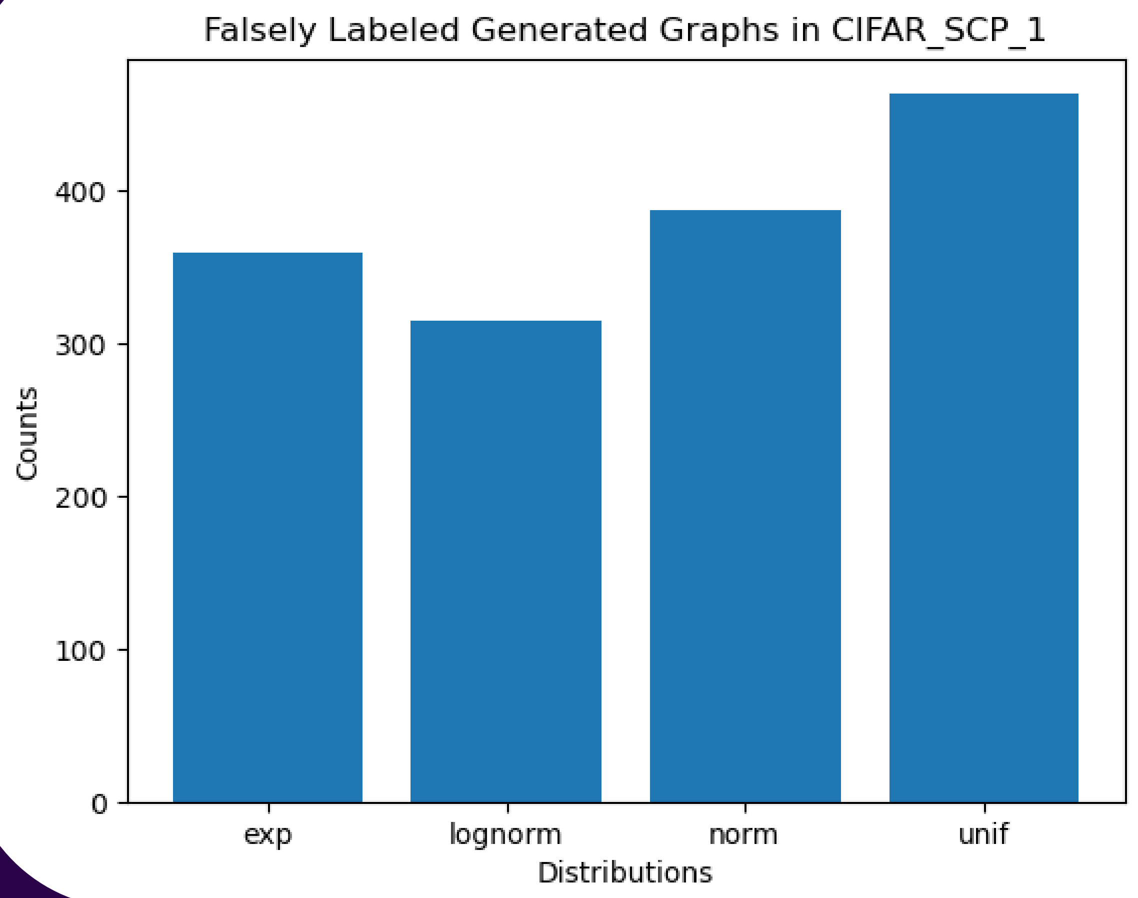
Scraped Graphs in CIFAR_GEN_1

Prediction	Count
True (graph)	2886
False (natural)	4867

Accuracy: 37.2243%

When graphs from the scraped dataset are put into the CIFAR_GEN_1, it predicts most of them are natural images.

EVALUATIONS



What does this mean?

Graphs with a uniform distribution were most often labelled as a natural image.

MISCLASSIFICATION AS NATURAL

Misclassified Exp

exp_20

exp_202

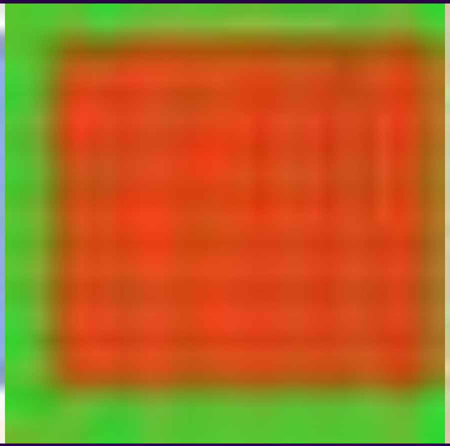
exp_392

exp_730

Misclassified Lognorm

lognorm_237

lognorm_544

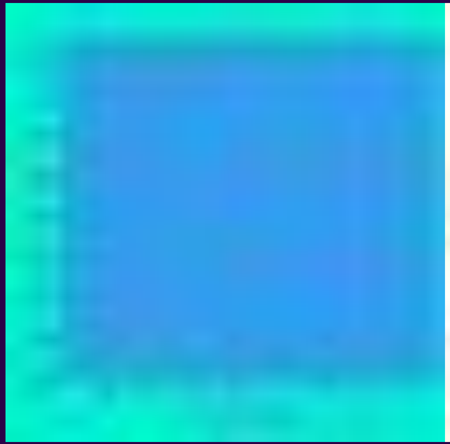


lornorm_918

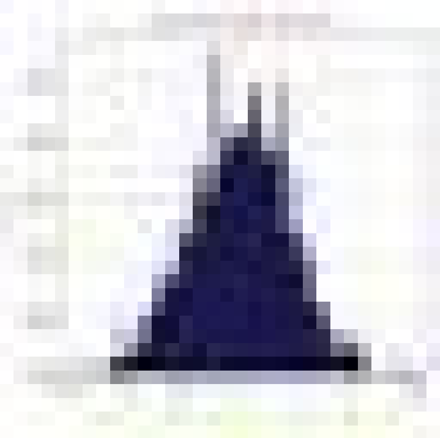
lognorm_1344

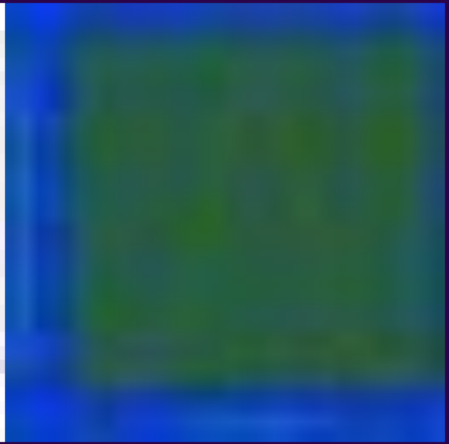
MISCLASSIFICATION AS NATURAL

Misclassified Norm

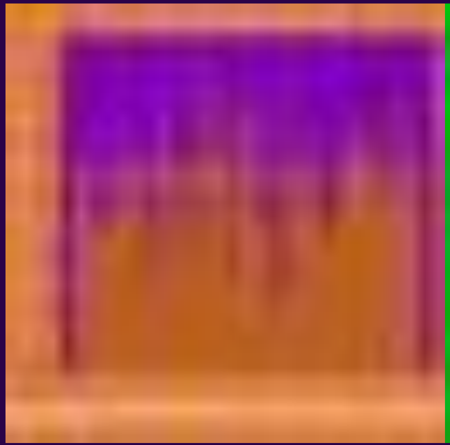

norm_399


norm_695

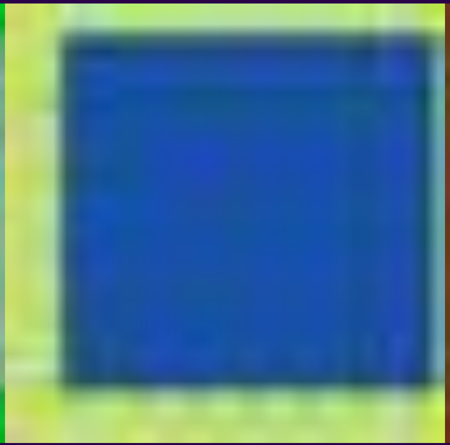

norm_1185


norm_1929

Misclassified Unif


unif_18


unif_352


unif_763

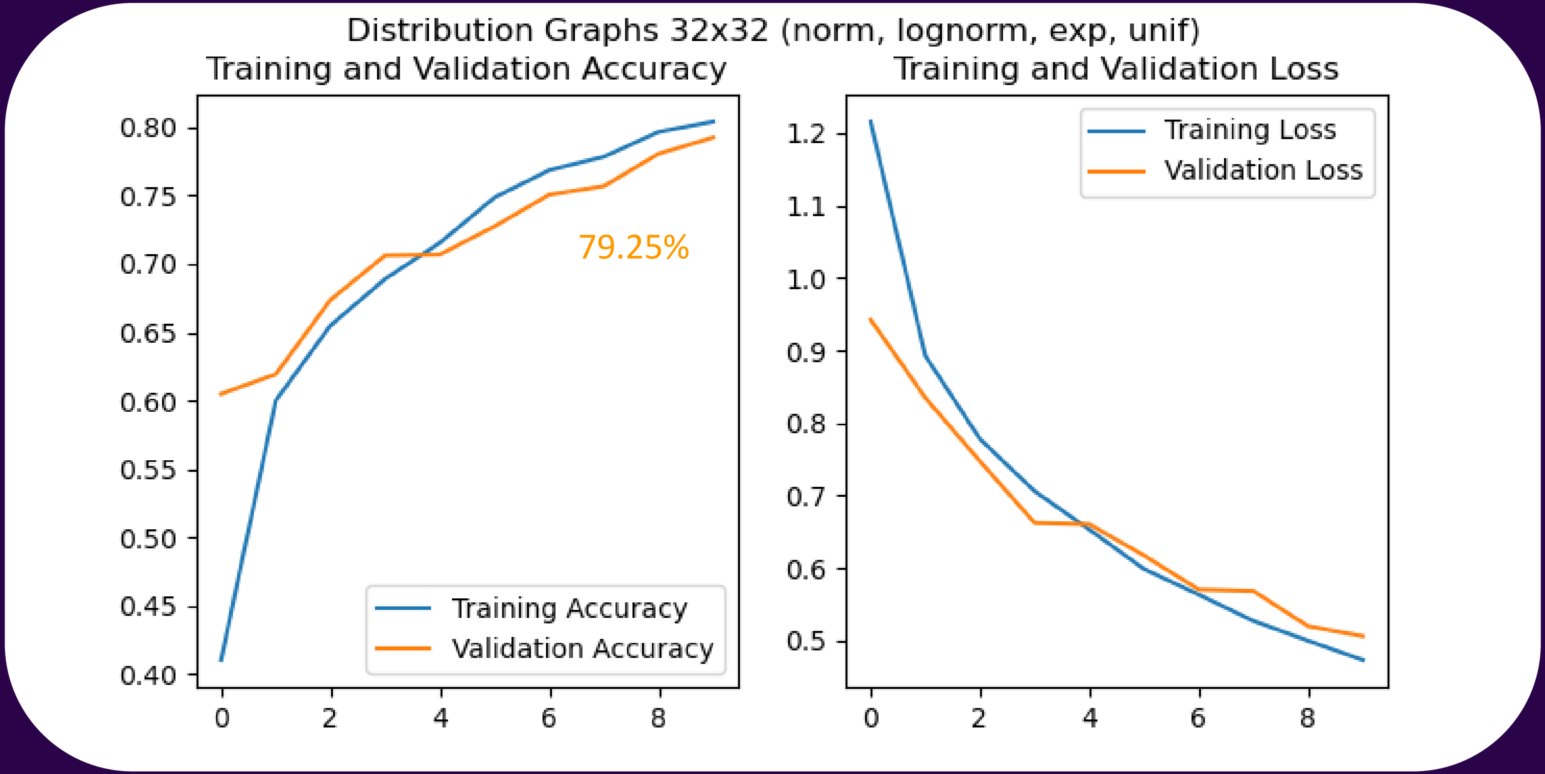

unif_1436

DISTRIBUTION GRAPH CLASSIFIERS

RESULTS

DIST_32X32_1

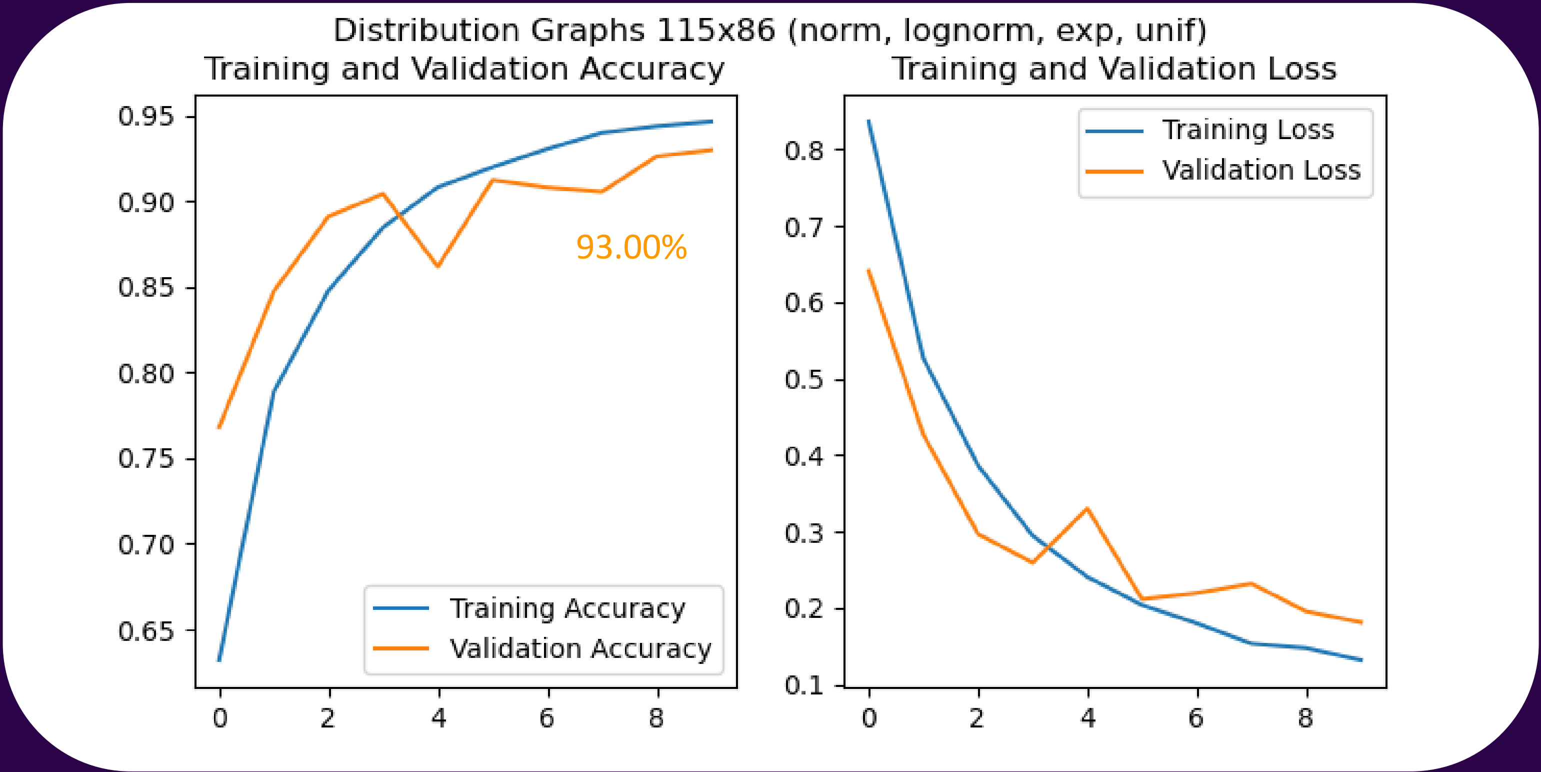
Distribution Graphs of Dimension 32x32



The model struggled more to classify the graphs, but still managed almost 80% validation accuracy after 10 epochs

DIST_115X86_1

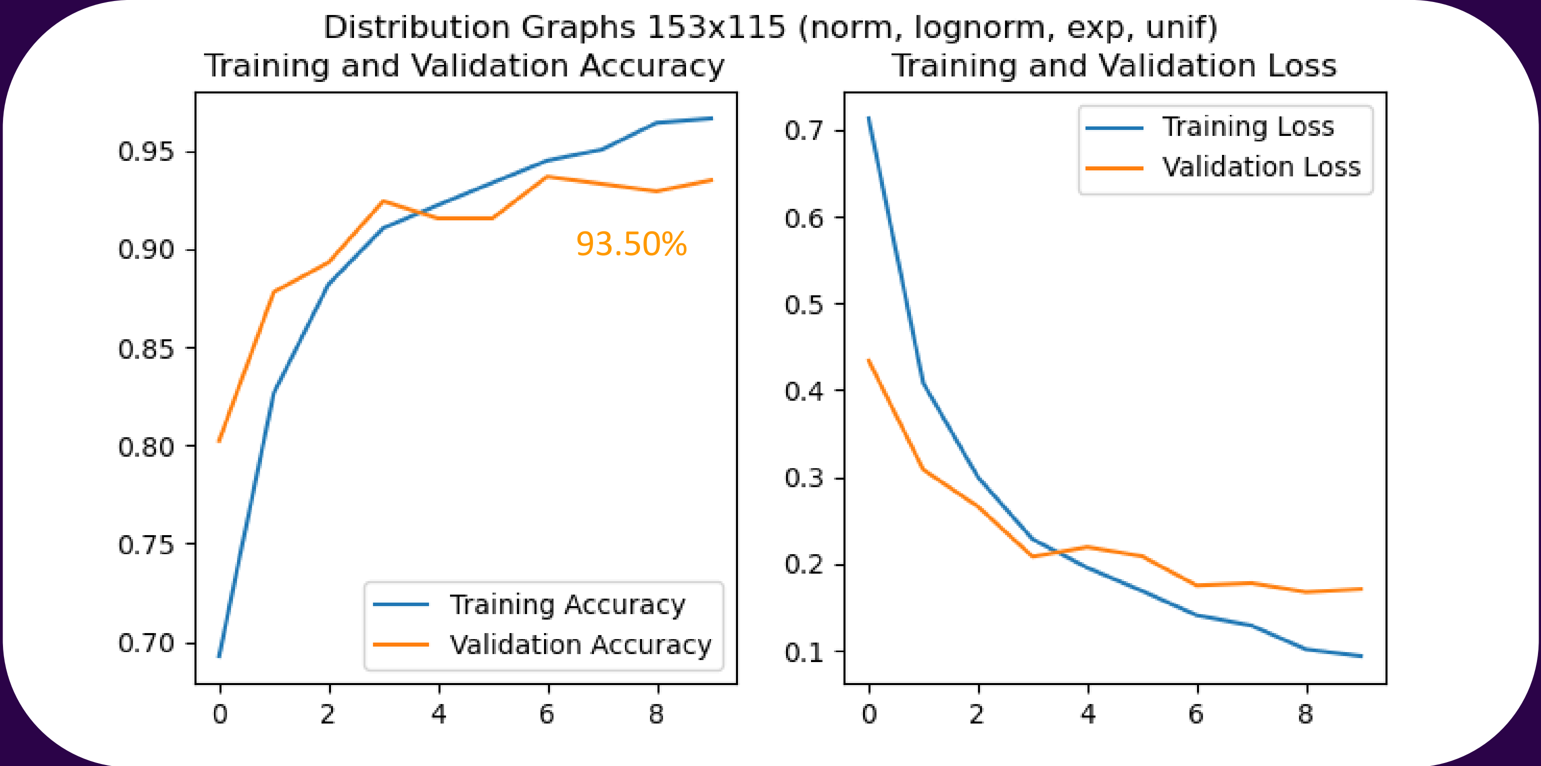
Distribution Graphs of Dimension 115x86



Increasing the quality of the image greatly increased accuracy

DIST_153X115_1

Distribution Graphs of Dimension 115x86



Diminishing return in regards of image quality since the validation accuracy only improved by 0.5% to 93.5% from the 115x86 images.

EVALUATION OVERVIEW

The model can't consistently classify ~7% of the images when validating. What are the images that are getting misclassified?

Accuracy is 96.63% with a total of 270 misclassifications

Exp in DIST_153x115_1

Prediction	Count
True (exp)	1851
False (other)	149

Accuracy: 92.55%

Lognorm in DIST_153x115_1

Prediction	Count
True (lognorm)	1925
False (other)	75

Accuracy: 96.25%

Norm in DIST_153x115_1

Prediction	Count
True (norm)	1960
False (other)	40

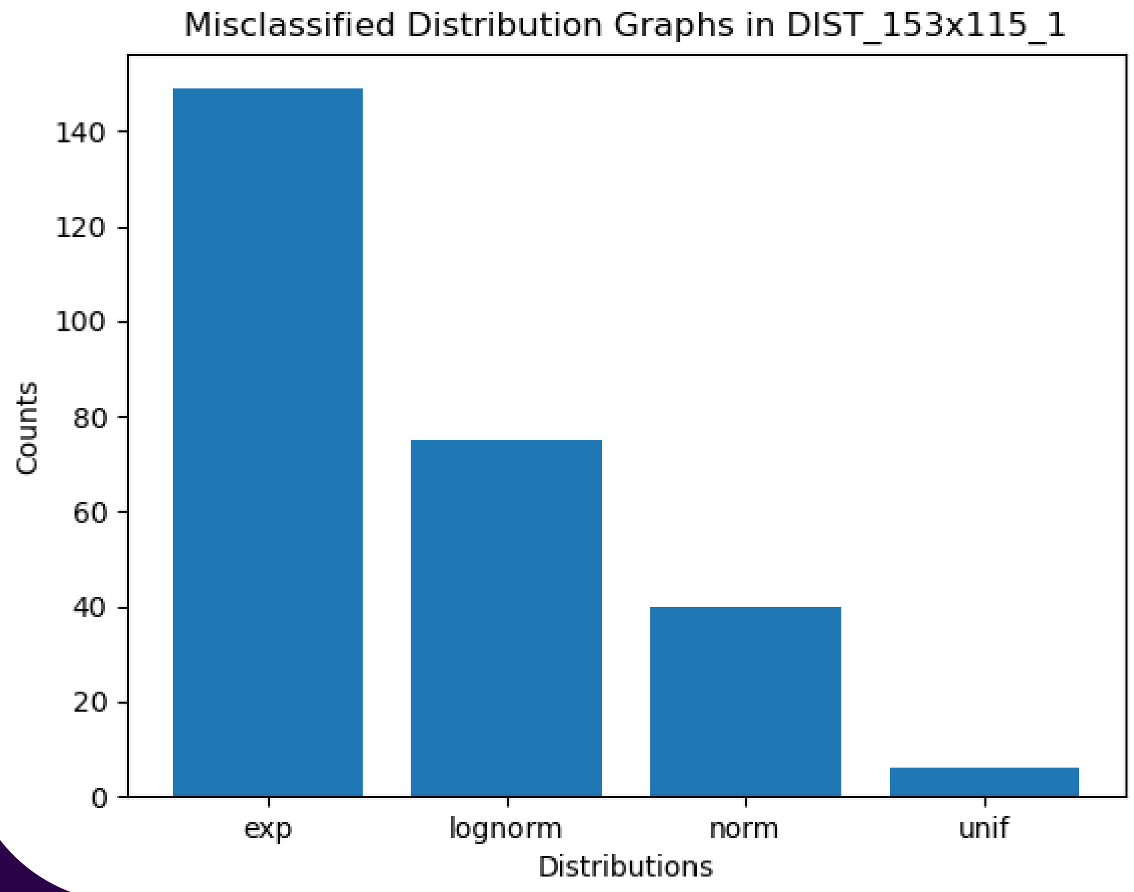
Accuracy: 98.00%

Unif in DIST_153x115_1

Prediction	Count
True (unif)	1994
False (other)	6

Accuracy: 99.70%

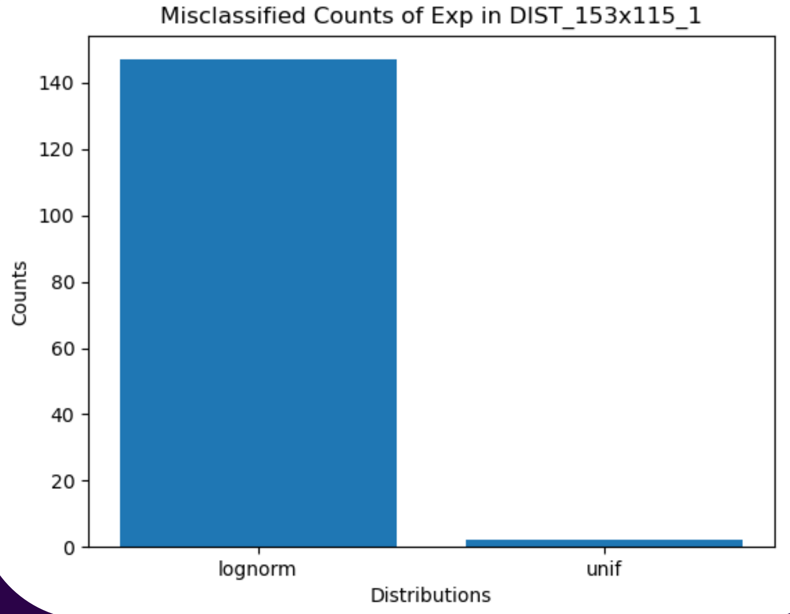
MISCLASSIFICATIONS



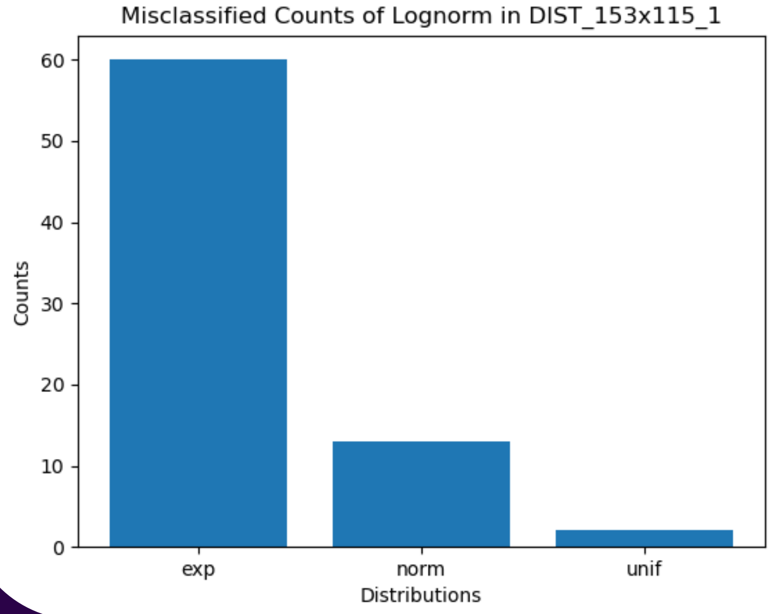
What does this mean?

Graphs with an exponential distribution were most often labelled incorrectly.

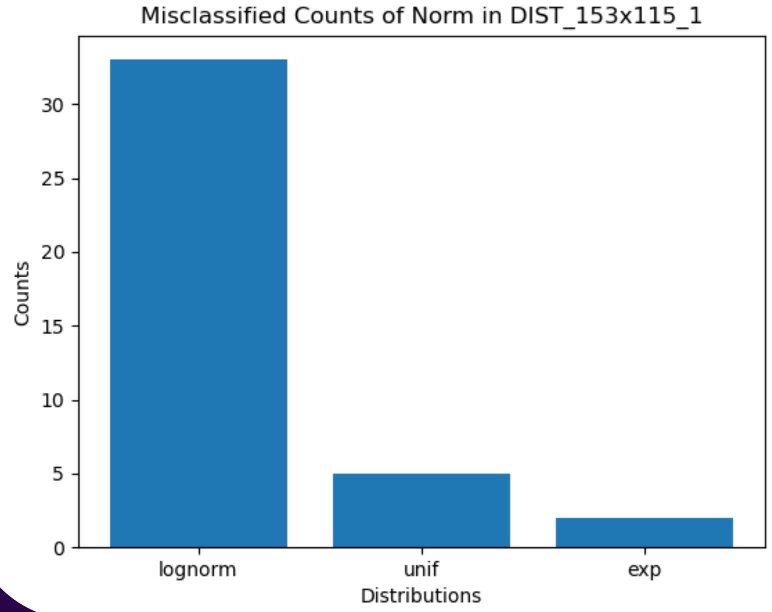
DEEPER DIVE



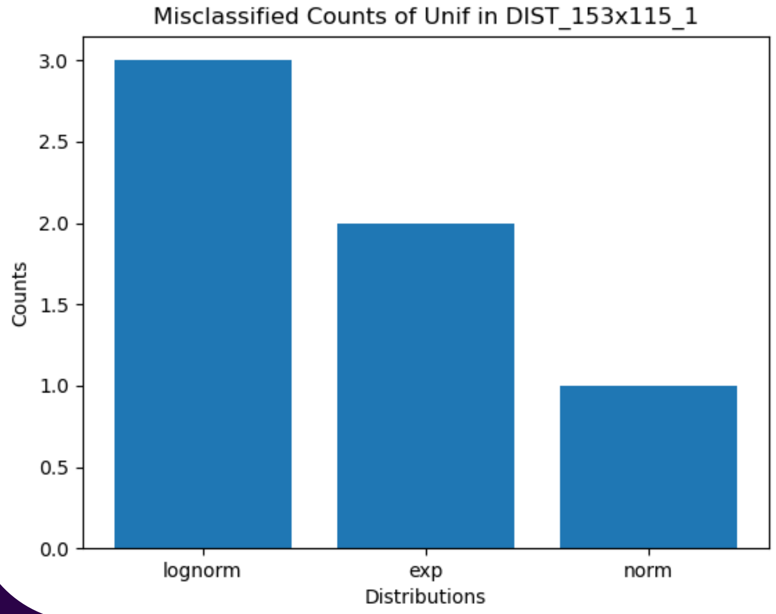
Misclassified Counts of Exp in DIST_153x115_1
Counts
0
20
40
60
80
100
120
140
lognorm
unif
Distributions



Misclassified Counts of Lognorm in DIST_153x115_1
Counts
0
10
20
30
40
50
60
exp
norm
unif
Distributions



Misclassified Counts of Norm in DIST_153x115_1
Counts
0
5
10
15
20
25
30
lognorm
unif
exp
Distributions



Misclassified Counts of Unif in DIST_153x115_1
Counts
0.0
0.5
1.0
1.5
2.0
2.5
3.0
lognorm
exp
norm
Distributions

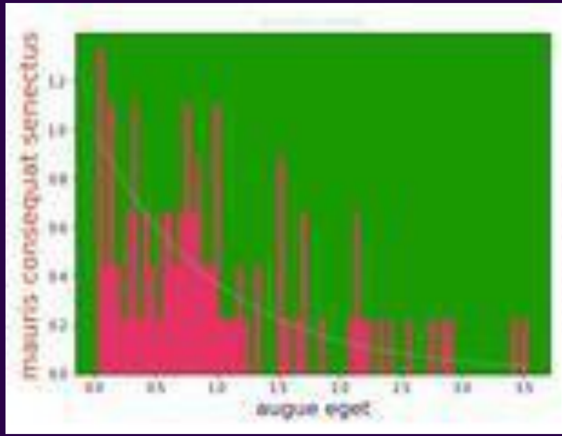
MISCLASSIFICATION RESULTS CSV

In order to understand the results better I created a CSV with relevant information.
Here is the misclassification of exponential distribution graphs as lognormal.

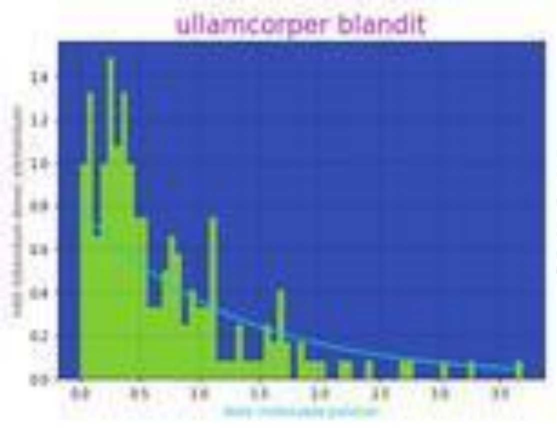
Ending prediction weights per class *Correct?* *File name* *Labeled based on max*

	exp	lognorm	norm	unif	prediction	file_name	max	label
27	5.698128	10.612542	-8.581626	-5.062102	False	exp_1021.jpg	10.612542	lognorm
454	7.196147	10.839309	-8.692248	-5.899051	False	exp_1406.jpg	10.839309	lognorm
1239	4.052152	9.712366	-6.139276	-5.969251	False	exp_313.jpg	9.712366	lognorm
1449	10.035354	11.493513	-8.789401	-13.515024	False	exp_502.jpg	11.493513	lognorm
1532	8.827777	9.538890	-9.788555	-8.289847	False	exp_578.jpg	9.538890	lognorm
1695	12.398116	12.656148	-8.937104	-15.547202	False	exp_724.jpg	12.656148	lognorm
1871	5.779149	11.178201	-6.324277	-9.075588	False	exp_883.jpg	11.178201	lognorm

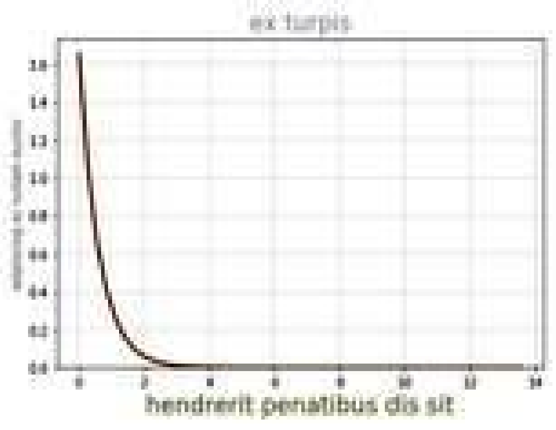
MISCLASSIFICATION OF EXP AS LOGNORM



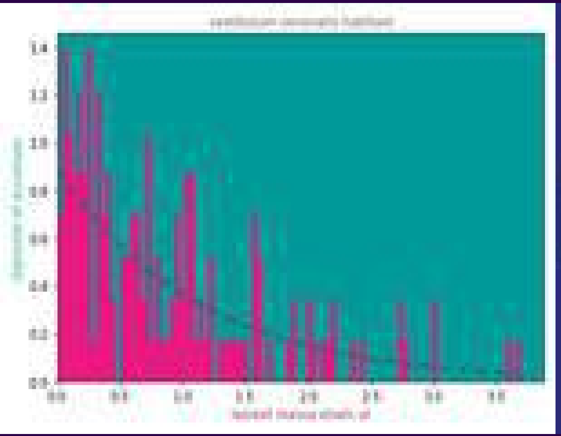
exp_313 = 9.7 prediction



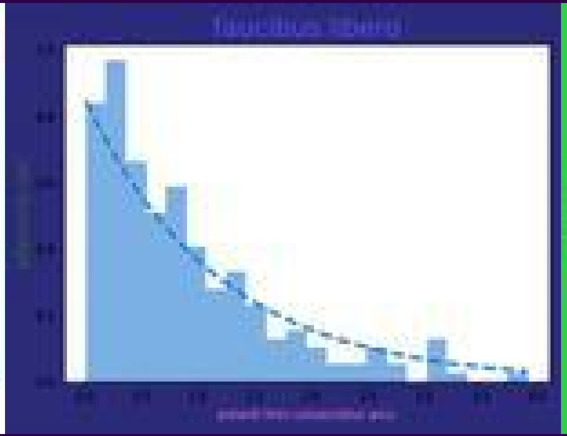
exp_502 = 11.5



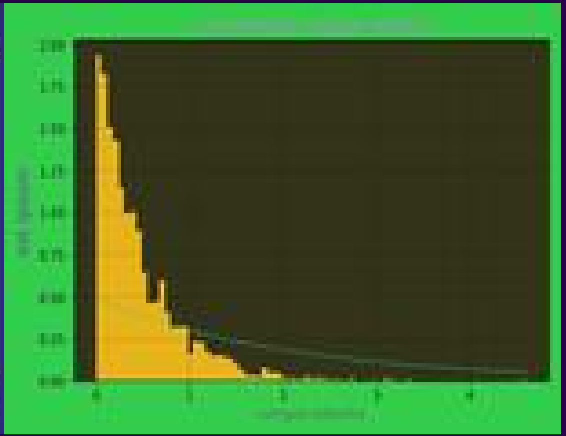
exp_578 = 9.5



exp_724 = 12.7

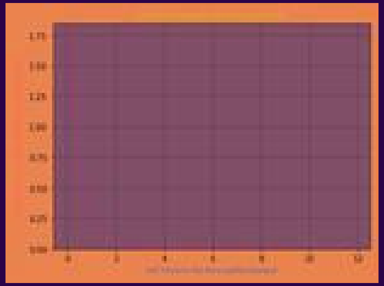


exp_883 = 11.2

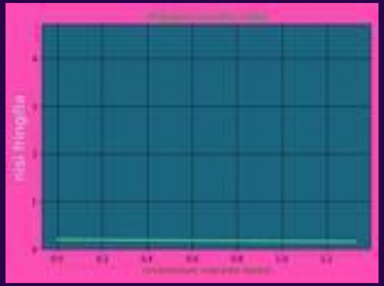


exp_1021 = 10.6

Bonus: Exp as Unif

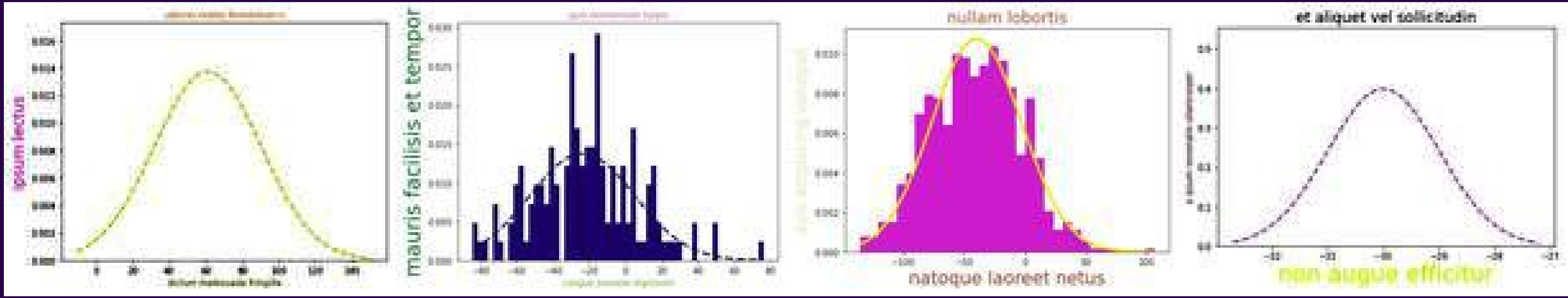


exp_1782



exp_1142

MISCLASSIFICATION OF NORM AS LOGNORM



norm_53 = 6.7 prediction

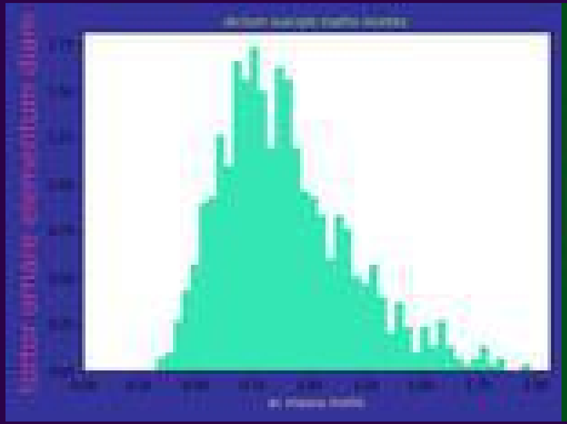
norm_193 = 8.9

norm_194 = 7.5

norm_1082 = 11.6

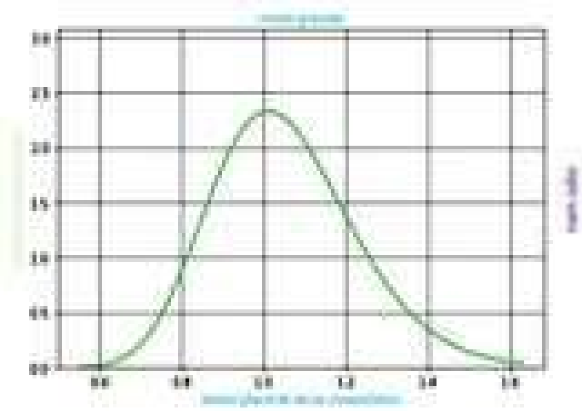
	exp	lognorm	norm	unif	prediction	file_name	max	label
94	-19.905157	11.556813	10.167370	-19.739372	False	norm_1082.jpg	11.556813	lognorm
1035	-5.109227	8.912791	7.967355	-30.035488	False	norm_193.jpg	8.912791	lognorm
1046	-12.689414	7.472771	7.200003	-14.969963	False	norm_194.jpg	7.472771	lognorm
1479	-9.045657	6.747765	6.157964	-17.828022	False	norm_53.jpg	6.747765	lognorm

MISCLASSIFICATION OF LOGNORM AS NORM

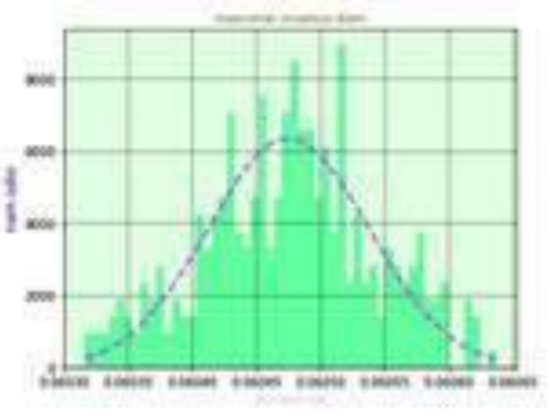


lognorm_520 = 9.1 prediction

lognorm_710 = 11.5



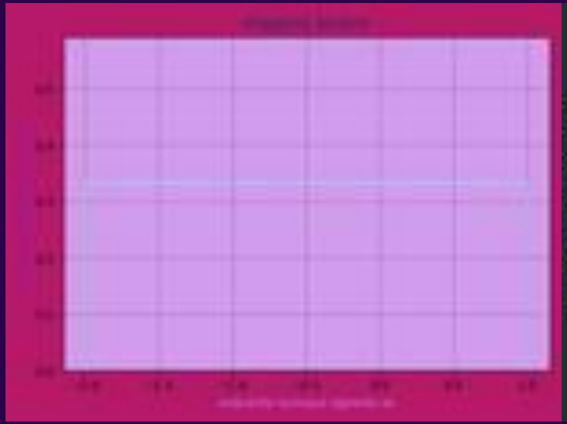
lognorm_972 = 6.2



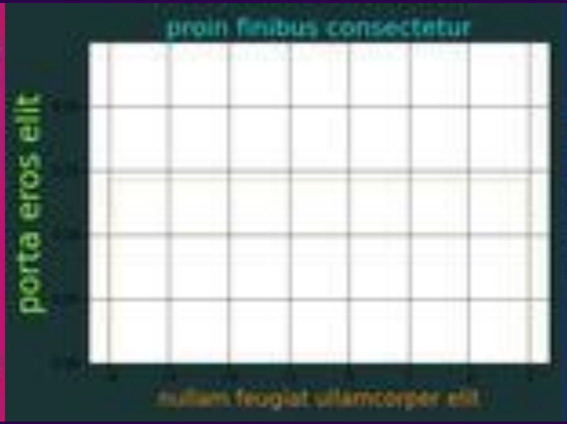
lognorm_1301 = 10.6

	exp	lognorm	norm	unif	prediction	file_name	max	label
338	-15.704831	6.110114	10.617505	-17.632048	False	lognorm_1301.jpg	10.617505	norm
1469	-15.871367	7.995641	9.133474	-16.884045	False	lognorm_520.jpg	9.133474	norm
1680	-27.291807	10.285934	11.453963	-20.160696	False	lognorm_710.jpg	11.453963	norm
1970	-12.753652	6.126523	6.247423	-9.829393	False	lognorm_972.jpg	6.247423	norm

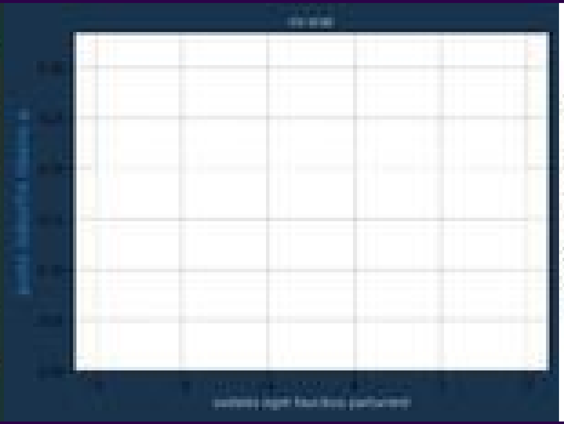
MISCLASSIFICATION OF UNIF



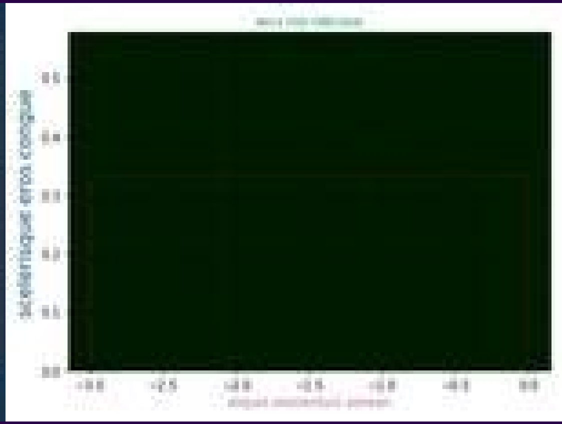
unif_451 as lognorm



unif_466 as norm



unif_609 as exp



unif_642 as exp

Not very surprising that these graphs are labeled seemingly randomly since they are mostly blank.
But it is impressive that there are only 6 misclassified unif graphs considering the amount that look like these.

	exp	lognorm	norm	unif	prediction	file_name	max	label
264	0.645309	1.522328	-3.437732	1.251064	False	unif_1235.jpg	1.522328	lognorm
1392	0.859410	1.038198	-3.107636	0.687651	False	unif_451.jpg	1.038198	lognorm
1408	-7.101572	-2.888915	1.196919	0.934827	False	unif_466.jpg	1.196919	norm
1567	1.126755	-0.254811	-3.411243	0.732441	False	unif_609.jpg	1.126755	exp
1604	3.546756	2.960398	-5.525770	-0.322343	False	unif_642.jpg	3.546756	exp
1769	-0.240980	0.551534	-2.195274	0.304812	False	unif_791.jpg	0.551534	lognorm

FINAL TEST EVALUATION

In order to truly evaluate my generated graphs and models, I needed to use graphs that were not used before.

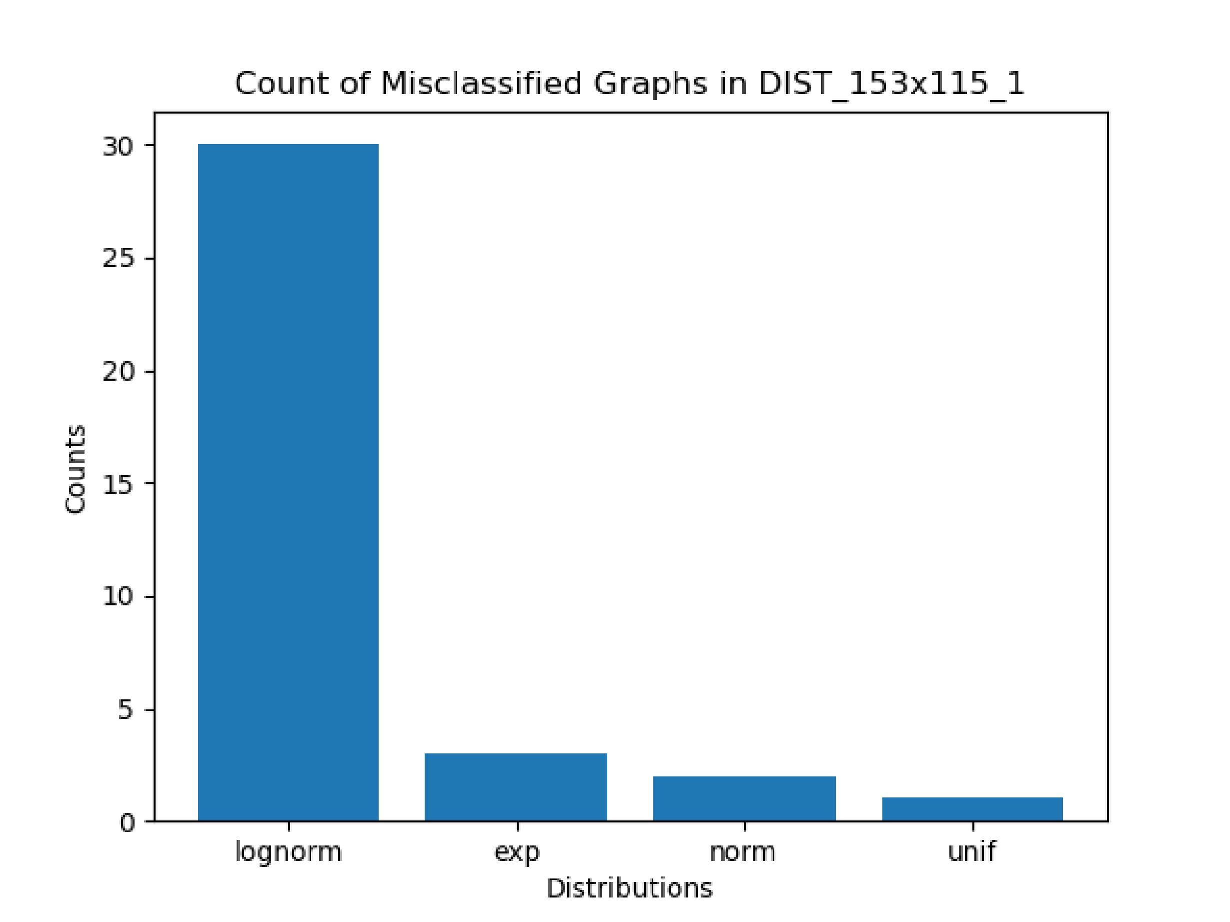
Exp	20 (new scraped) + 1 (hand-drawn)
Lognorm	20 + 1
Norm	20 + 1
Unif	20 + 1

Test images in DIST_153x115_1

Prediction	Count
True	52
False	36

Accuracy: 59.091%

Best case is 100% accuracy



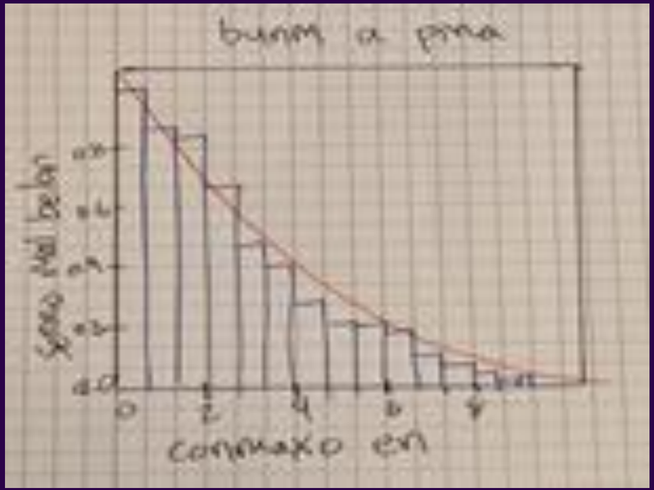
Most incorrectly classified graphs were misclassified as lognorm

FINAL TEST EVALUATION

Misclassified Exp Graphs

40% of exp graphs were misclassified as lognorm

	exp	lognorm	norm	unif	max	file_name	actual	label	prediction
4	-4.166146	8.161812	0.978815	-10.229221	8.161812	exp_12.jpeg	exp	lognorm	False
12	3.851117	4.496506	-3.405139	-5.299541	4.496506	exp_2.jpeg	exp	lognorm	False
13	5.872565	11.971514	-6.275291	-12.128517	11.971514	exp_20.jpeg	exp	lognorm	False
14	9.207311	10.764248	-11.954131	-5.385407	10.764248	exp_3.jpeg	exp	lognorm	False
16	-5.917483	15.367903	-4.724207	-5.200668	15.367903	exp_5.jpeg	exp	lognorm	False
19	6.133784	6.400875	-5.021023	-7.424274	6.400875	exp_8.jpeg	exp	lognorm	False
20	0.511647	8.585008	-3.287844	-6.293257	8.585008	exp_9.jpeg	exp	lognorm	False
21	-2.766488	8.546496	0.333981	-9.835137	8.546496	exp_hand_0.jpeg	exp	lognorm	False



exp_hand_0

Misclassified Lognorm Graphs

	exp	lognorm	norm	unif	max	file_name	actual	label	prediction
22	-13.257631	5.875923	-2.456502	7.38458	7.384580	lognorm_0.jpeg	lognorm	unif	False
38	3.730291	2.013594	-0.048096	-11.51519	3.730291	lognorm_5.jpeg	lognorm	exp	False

Only two misclassifications of lognorm graphs

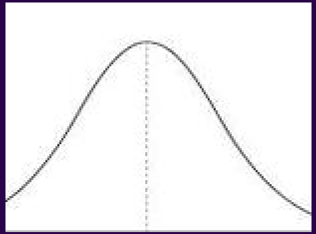
FINAL TEST EVALUATION

Misclassified

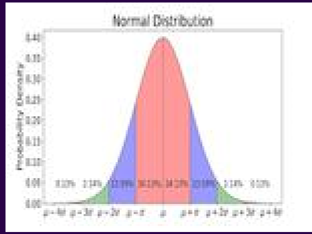
	exp	lognorm	norm	unif	max	file_name	actual	label	prediction
45	-14.977435	3.678837	1.503011	1.734718	3.678837	norm_1.jpeg	norm	lognorm	False
47	-0.316338	5.540563	1.644825	-18.002310	5.540563	norm_11.jpeg	norm	lognorm	False
49	-14.409090	10.438942	7.760438	-26.327517	10.438942	norm_13.jpeg	norm	lognorm	False
50	-5.349610	7.274977	1.170779	-6.384265	7.274977	norm_14.jpeg	norm	lognorm	False
51	-6.992462	7.189061	4.943288	-15.288852	7.189061	norm_15.jpeg	norm	lognorm	False
55	-4.783489	4.332031	3.838747	-14.778937	4.332031	norm_19.jpeg	norm	lognorm	False
59	-23.318886	12.883771	-0.864109	5.143599	12.883771	norm_4.jpeg	norm	lognorm	False
63	-25.816198	16.998684	4.591496	-9.728744	16.998684	norm_8.jpeg	norm	lognorm	False
64	-6.611537	7.848427	0.576409	-4.260340	7.848427	norm_9.jpeg	norm	lognorm	False

Correctly Predicted

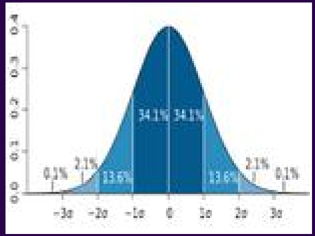
	exp	lognorm	norm	unif	max	file_name	actual	label	prediction
44	-7.893614	-0.224976	5.218153	-7.976728	5.218153	norm_0.jpeg	norm	norm	True
46	-39.708008	11.495808	21.575079	-34.096115	21.575079	norm_10.jpeg	norm	norm	True
48	-21.136816	4.670593	12.121604	-19.625480	12.121604	norm_12.jpeg	norm	norm	True
52	-18.647070	7.492292	9.302481	-18.890394	9.302481	norm_16.jpeg	norm	norm	True
53	-14.246734	3.805575	3.969000	-3.098505	3.969000	norm_17.jpeg	norm	norm	True
54	-14.695104	3.837751	9.268262	-14.621681	9.268262	norm_18.jpeg	norm	norm	True
56	-23.055544	8.824812	14.742339	-26.035667	14.742339	norm_2.jpeg	norm	norm	True
57	-15.316313	1.433591	9.032508	-11.850796	9.032508	norm_20.jpeg	norm	norm	True
58	-12.434736	1.432740	2.071673	1.228879	2.071673	norm_3.jpeg	norm	norm	True
60	-16.548819	4.050271	8.853429	-9.732455	8.853429	norm_5.jpeg	norm	norm	True
61	-22.864944	9.062793	9.793877	-9.050050	9.793877	norm_6.jpeg	norm	norm	True
62	-31.107384	11.258694	11.843807	-7.342812	11.843807	norm_7.jpeg	norm	norm	True
65	-16.542665	8.398274	13.159173	-28.154154	13.159173	norm_hand_0.jpeg	norm	norm	True



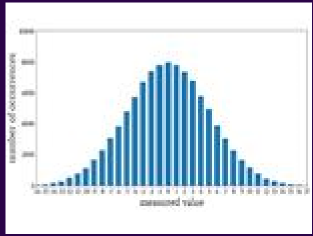
norm_8



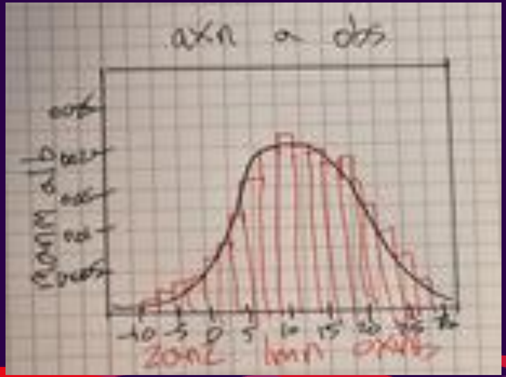
norm_11



norm_13



norm_15



norm_hand_0

True

Norm = 13.159

Lognorm = 8.398

FINAL TEST EVALUATION

Misclassified Unif Graphs

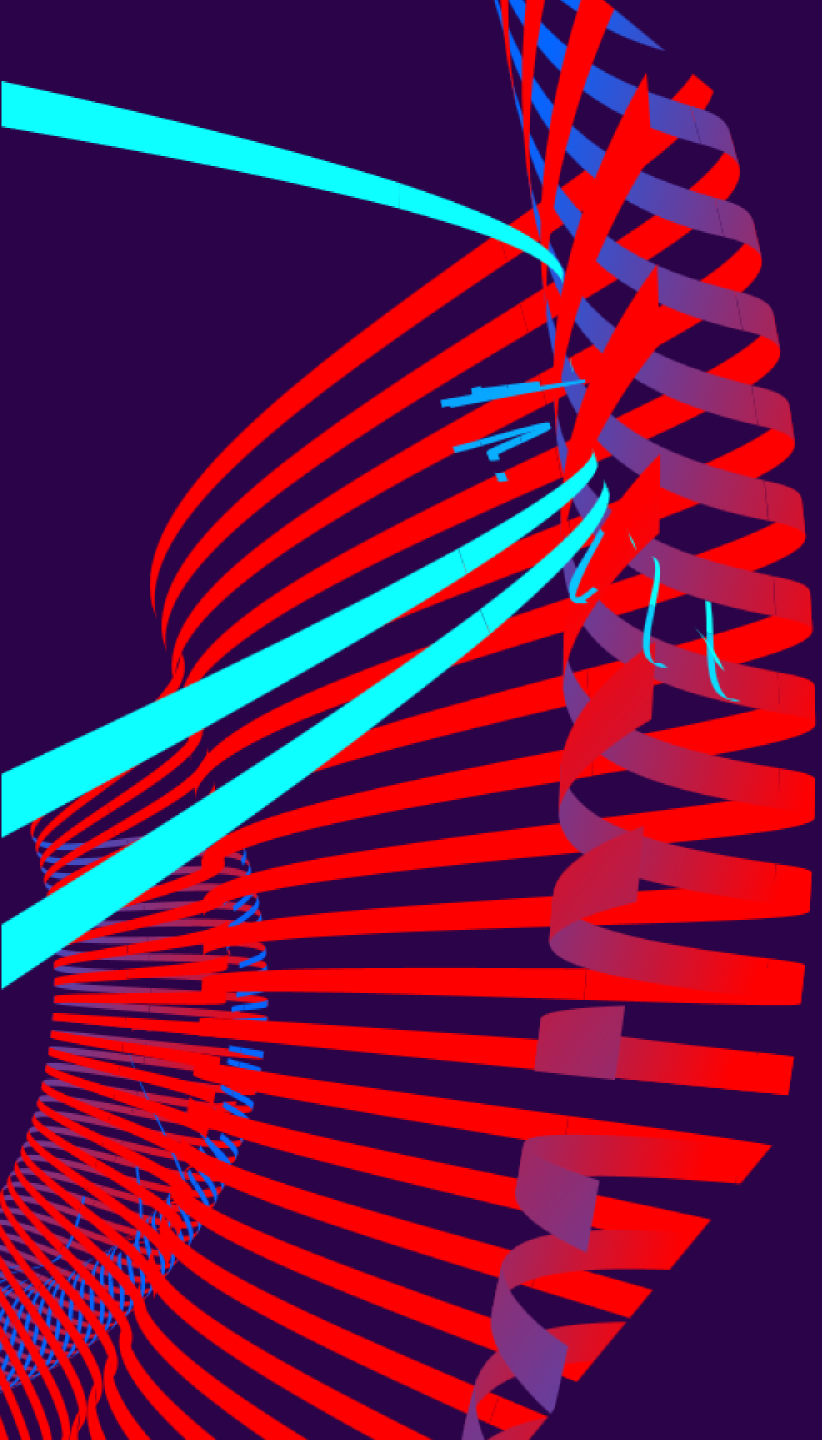
	exp	lognorm	norm	unif	max	file_name	actual	label	prediction
66	-0.709542	1.184137	-0.535988	-1.623665	1.184137	unif_0.jpeg	unif	lognorm	False
67	-4.572587	5.335835	0.411009	-3.925213	5.335835	unif_1.jpeg	unif	lognorm	False
68	-2.482021	-0.051051	2.781942	-4.327006	2.781942	unif_10.jpeg	unif	norm	False
69	-4.820385	9.742123	-7.255406	6.477165	9.742123	unif_11.jpeg	unif	lognorm	False
70	-0.560627	6.147130	-2.217081	-3.020845	6.147130	unif_12.jpeg	unif	lognorm	False
71	1.115218	1.182753	-0.670932	-5.319949	1.182753	unif_13.jpeg	unif	lognorm	False
72	-0.196241	7.653762	-2.495958	-3.569571	7.653762	unif_14.jpeg	unif	lognorm	False
74	-0.469846	5.252478	-4.977860	1.503776	5.252478	unif_16.jpeg	unif	lognorm	False
75	-0.203208	4.199161	1.574089	-10.120164	4.199161	unif_17.jpeg	unif	lognorm	False
78	-2.388467	3.969737	-0.653423	-2.820281	3.969737	unif_2.jpeg	unif	lognorm	False
80	1.630678	6.316803	-0.730518	-8.067163	6.316803	unif_3.jpeg	unif	lognorm	False
81	-5.665611	4.946749	1.818700	-6.484777	4.946749	unif_4.jpeg	unif	lognorm	False
82	-4.420216	3.025204	1.048803	-2.055534	3.025204	unif_5.jpeg	unif	lognorm	False
83	5.331371	4.629284	-3.963360	-6.628367	5.331371	unif_6.jpeg	unif	exp	False
85	-4.064364	4.501654	-3.487114	3.328222	4.501654	unif_8.jpeg	unif	lognorm	False
86	-6.592189	-0.654820	2.711003	-1.703876	2.711003	unif_9.jpeg	unif	norm	False
87	0.534510	0.080019	0.436011	-6.152885	0.534510	unif_hand_0.jpeg	unif	exp	False

It was surprising how 61.90% of unif graphs in the test dataset were misclassified as lognorm since the generated unif graphs were predicted so accurately.

Unif in DIST_153x115_1

Prediction	Count
True (unif)	1994
False (other)	6

Accuracy: 99.70%



CONCLUSION

CONCLUSION

Overall Goals Accomplished

- can successfully identify graphs from natural images with greater than ~98% accuracy
- can successfully identify graphs with different distribution types with ~93% accuracy

Some Surprising Results

- scraped graphs in CIFAR_GEN_1 are misclassified as natural images ~63% of the time
  - This enforces my idea that many of the images in SCP were more ‘graph-like’ and not graphs
- unif in DIST_153x115_1 were classified accurately ~99% of the time, but 20% of unif graphs in CIFAR_SCP_1 were misclassified as natural images
  - Graphs with the uniform distribution were very unique from the other distribution graphs but not very unique from natural images

CONCLUSION

More Preprocessing Considerations to Improve Results

- tf.keras.layers.RandomBrightness()
- tf.keras.layers.RandomContrast()
- tf.keras.layers.RandomCrop()
- tf.keras.layers.RandomFlip()
- tf.keras.layers.RandomRotation()

```
random_bright = tf.keras.layers.RandomBrightness(factor=0.2)

# An image with shape [2, 2, 3]
image = [[[1, 2, 3], [4 ,5 ,6]], [[7, 8, 9], [10, 11, 12]]]

# Assume we randomly select the factor to be 0.1, then it will apply
# 0.1 * 255 to all the channel
output = random_bright(image, training=True)

# output will be int64 with 25.5 added to each channel and round down.
tf.Tensor([[[26.5, 27.5, 28.5]
            [29.5, 30.5, 31.5]]
           [[32.5, 33.5, 34.5]
            [35.5, 36.5, 37.5]]],
          shape=(2, 2, 3), dtype=int64)
```

```
model = tf.keras.Sequential([
  layers.Rescaling(1./255,
  # add more randomness to images after rescaling
])
```

https://www.tensorflow.org/api_docs/python/tf/image/random_brightness

CONCLUSION

Another common CNN architecture is to stack two convolutional layers before each pooling layer, as illustrated in Figure 5. This is strongly recommended as stacking multiple convolutional layers allows for more complex features of the input vector to be selected.

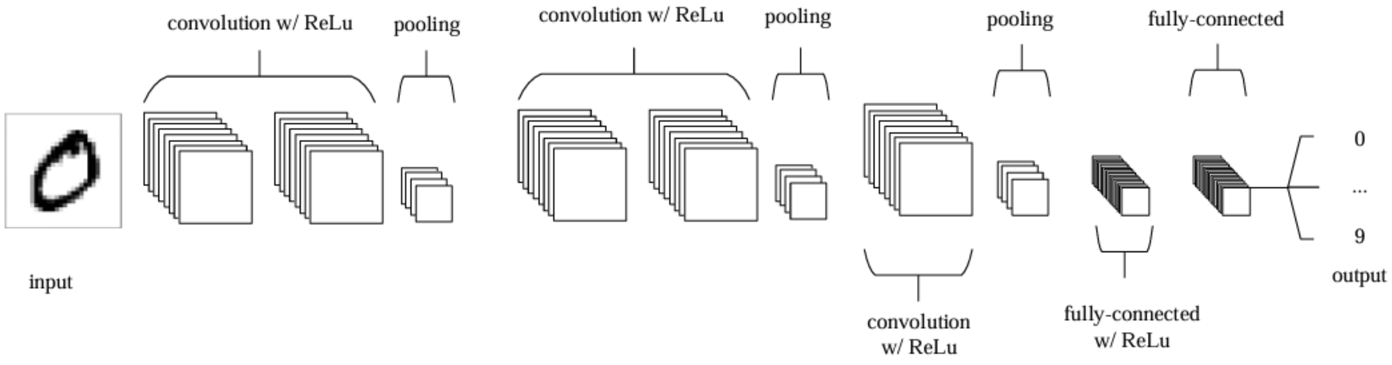


Fig. 5: A common form of CNN architecture in which convolutional layers are stacked between ReLus continuously before being passed through the pooling layer, before going between one or many fully connected ReLus.

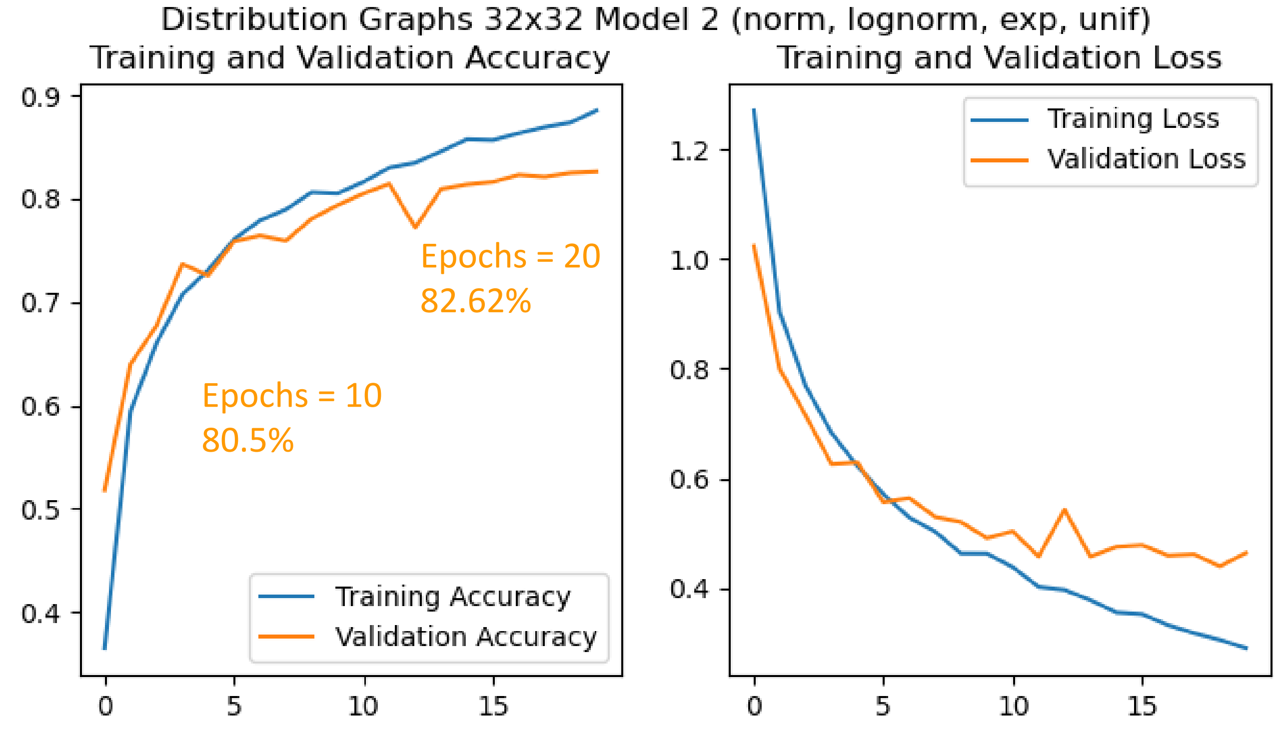
[O'Shea et al.]

Further Research and Improvement

- Change model structure
- Use more images to train models
- Improve generated graphs
- Add more distributions
- Use entire reports as input and identify pages with graphs on them

DIST_32X32_2

Distribution Graphs of Dimension 32x32



By doubling the number of filters in the convolution layers to see minor improvements after but we start to run into the issue of overfitting

79.25% for DIST_32x32_1
After 10 Epochs.

REFERENCES (1/4)

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., . . . Google Brain. (2016). TensorFlow: A system for large-scale machine learning. 12th USENIX Symposium on Operating Systems Design and Implementation (pp. 265-283). Savannah, GA, USA: USENIX Association.

Agarap, A. F. (2019). Deep Learning using Rectified Linear Units (ReLU). San Diego: 3rd International Conference for Learning Representations. Retrieved from https://arxiv.org/abs/1803.08375

Bishop, C. M. (1995). Neural Networks for Pattern Recognition. Oxford: Clarendon Press. Retrieved from https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf

Brownlee, J. (2019, July 05). A Gentle Introduction to Pooling Layers for Convolutional Neural Networks. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

Clark, A. (2015). Pillow (PIL Fork) Documentation. readthedocs. Retrieved from https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf

REFERENCES (2/4)

CS231n. (2023). (Stanford University) Retrieved January 2024, from Convolutional Neural Networks for Visual Recognition: https://cs231n.github.io/convolutional-networks/

Giskard. (n.d.). Rectified Linear Unit (ReLU). Retrieved from Giskard: https://www.giskard.ai/glossary/rectified-linear-unit-relu

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., … Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585, 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90–95.

Kamakshi, V., & Krishnan, N. C. (2023, August 1). Explainable Image Classification: The Journey So Far and the Road Ahead. 4, 620–651. Retrieved from https://doi.org/10.3390/ai4030033

Kingma, D. P., & Ba, J. L. (2015). Adam: A Method for Stochastic Optimization. ICLR. Retrieved from https://arxiv.org/abs/1412.6980

REFERENCES (3/4)

Krizhevsky, A., Nair, V., & Hinton, G. (2009). The CIFAR-10 dataset. Retrieved from Canadian Institute For Advanced Research: https://www.cs.toronto.edu/~kriz/cifar.html

Kumar, P., & Codicals. (2021, August 24). Max Pooling, Why use it and its advantages. Retrieved from Medium: https://medium.com/geekculture/max-pooling-why-use-it-and-its-advantages-5807a0190459

Li, L. (2018, March 23). Data Visualization. Retrieved from Medium: https://medium.com/@Lynia_Li/as-you-know-there-are-many-types-of-charts-to-be-used-in-data-visualization-54da9b97092e

McDonald's. (2022). 2022 Annual Report. Retrieved from https://corporate.mcdonalds.com/content/dam/sites/corp/nfl/pdf/MCD_2023_Annual_Report.pdf

McKinney, W., & others. (2010). Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51–56).

O'Shea, K., & Nash, R. (2015). An Introduction to Convolutional Neural Networks. Aberystwyth University. ResearchGate. Retrieved from: https://doi.org/10.48550/arXiv.1511.08458

REFERENCES (4/4)

SandhyaKrishnan02. (2023). An overview of probability distribution. Retrieved from Kaggle: https://www.kaggle.com/discussions/general/366492

SunEdition. (2021). Graphs Dataset. Retrieved from Kaggle: https://www.kaggle.com/datasets/sunedition/graphs-dataset

The 365 Team. (2023, June 15). What Is Cross-Entropy Loss Function? Retrieved from 365DataScience: https://365datascience.com/tutorials/machine-learning-tutorials/cross-entropy-loss/

THANK YOU

Patrick Spohr

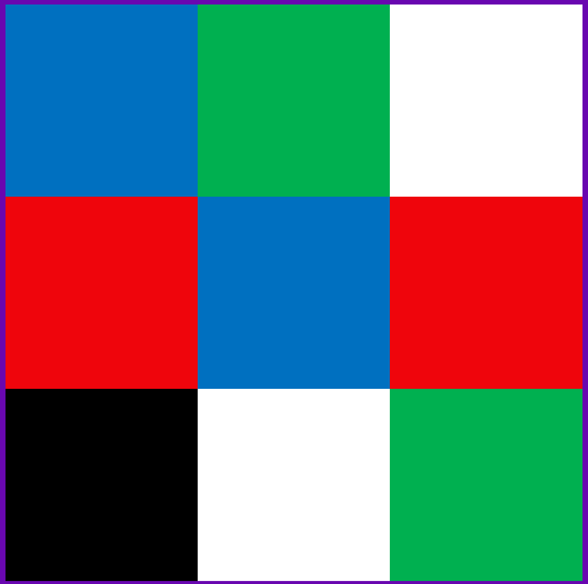
HTW Berlin – FAR Master‘s

Deep Learning Seminar

Prof Dr Alla Petukhina

ANSWER

Image Quiz



(((0, 0, 255), (0, 255, 0), (255, 255, 255)),
((255, 0, 0), (0, 0, 255), (255, 0 , 0)),
((0, 0, 0), (255, 255, 255), (0, 255, 0)))

(3, 3, 3)