Image Classification with CNNs

An IRONHACK project

```
def Group_3():
   [‘Diego’, ‘Javi’, ‘Salva’]
```

1 .- GROUP NUMBER & MEMBERS

Javier
Ansoleaga

Salva
Vento Asturias

Diego
Alonso

GROUP 3

2.- CHOSEN DATASET

CIFAR-10

Why?

1. We were **familiar** with the dataset as it was the one we used.
2. It offers more **variety** than Animals 10 from cars to frogs which will offer a more diverse output for the model.
3. It is a **balanced** dataset as their classes are equally distributed.
4. As their images are small it allows **fast prototyping** and hyperparameter **tuning.**
5. The size of the images also grants a **faster training** and the possibility of having **quicker transfer learning.**

At the end we chose CIFAR 10 for image because it has a good balance between simplicity, versatility and the challenge. It allows for fast iteration, experimentation with different models, and it is efficient while training models.

3.- PROBLEM OVERIVIEW

The main problems that this dataset could throw at our model will be related to dataset size, low resolution of images and inter-class similarity:

PROBLEMS

1. As the images have a low resolution (32x32px) this may lead to **feature misclassification** (like small facial features on animals) and can be difficult for the model also may lead to errors when trying to **differentiate among similar classes** e.g., cats v. dogs.
2. Due to both limited dataset size and low-res images the model will achieve **good scores in training** will probably present **problems with real-world images** classification.
3. Baring inter-class similarity and size (images & dataset) the model will certainly **struggle to obtain good accuracy scores in similar classes** such as cars and trucks or dogs and cats.

SOLUTIONS TO BE IMPLEMENTED

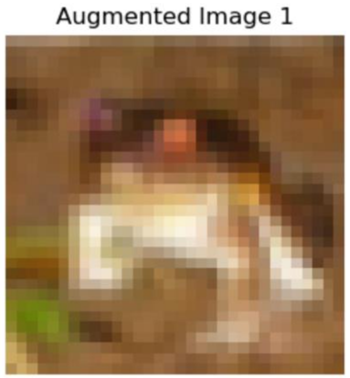
1. Balance testing and training accuracy by **tweaking** hyperparameters, dropout, layer optimization etc.
2. Try data **augmentation techniques** to increase the dataset for training purposes and use **class weighting** to reduce inter-class similarity.

APPROACH: Start with simple CNNs and gradually increase complexity.

4.- DATA PREPROCESING

1. **Load images** from the CIFAR 10 dataset.
2. **Plot a 10x10 grid** to see what the dataset looks like.
3. **Rescale / normalize** the data rescaling pixel values to [0, 1].
4. **Convert labels to** categorical (One-Hot Encoding).
5. **Rename labels** from 0-1 with class names dog, cat, frog…
6. **Splitting Data** into Training and Validation Sets
7. **Data augmentation** firstly applied to the whole dataset then as layers on the model.

   Provided the code with # to easily activate and deactivate.



Zoom

Displace

Flip

Stretch

4.- DATA PREPROCESING

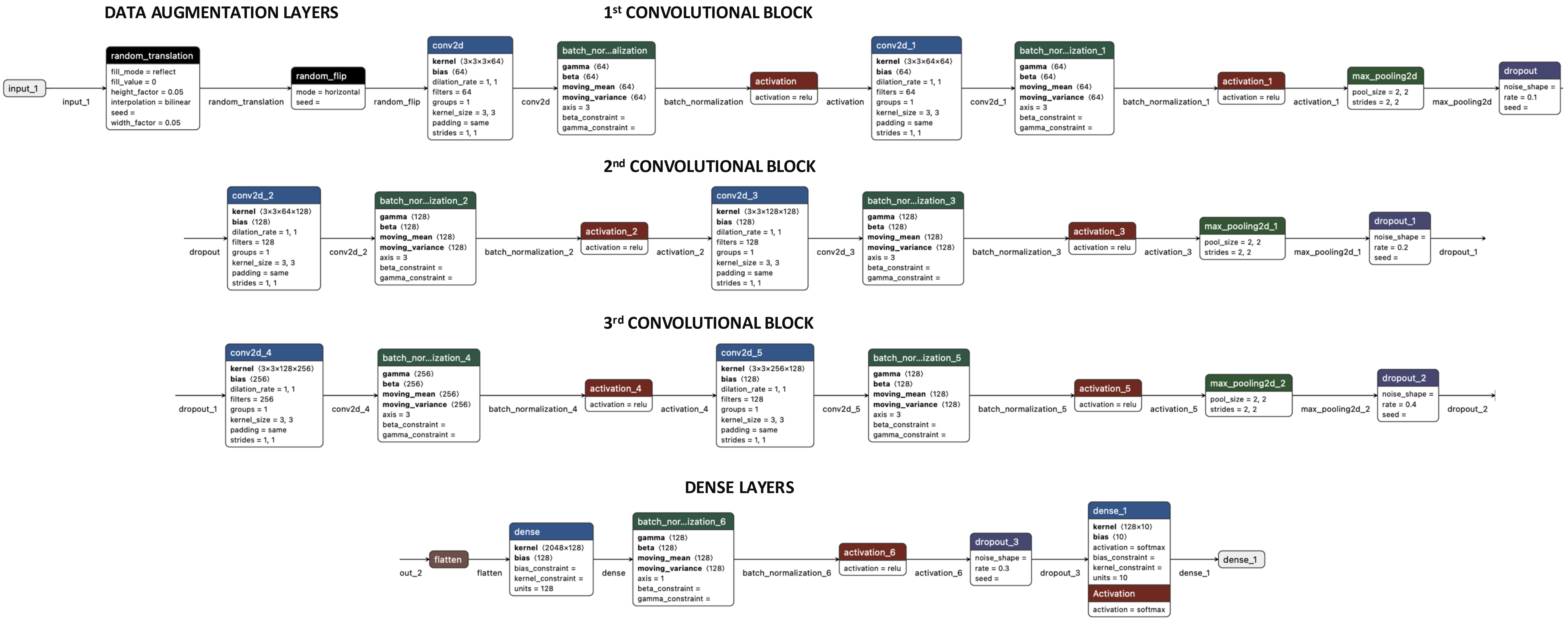
Things that we would have done if we had more time…

Specially after testing our best model there are a few things that might have been interesting to apply while processing the data such as:

1. Increase the **resolution** of the images using interpolation.
2. **Zero-Centering** the data (mean normalization).
3. **PCA** for noise and dimensionality reduction specially in combination with resolution scaling.

Probably increasing the image resolution would have a high-cost while in terms of computing but it would have surely increased the accuracy with real-world images.

5.- CNN ARCHITECTURE DESIGN



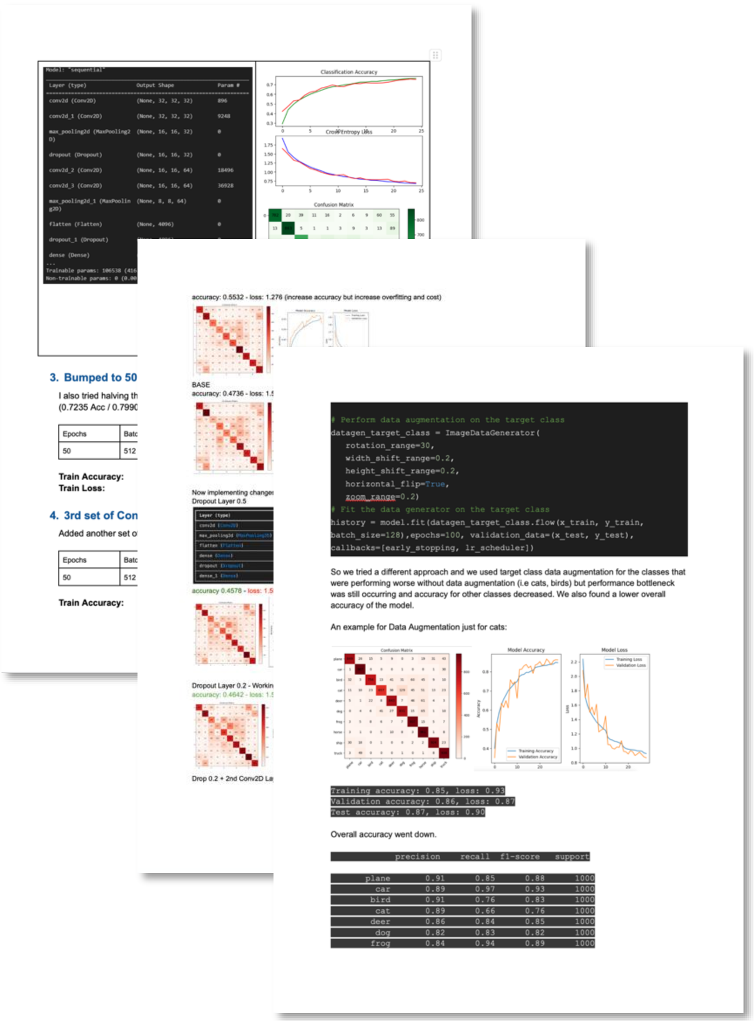
5.- CNN ARCHITECTURE DESIGN

```
# Defining the model
model = Sequential([
Input(shape=(32, 32, 3)),

# Data augmentation layers
RandomTranslation(height_factor=0.05, width_factor=0.05),
RandomFlip("horizontal"),

# 1st Convolutional Block
Conv2D(64, (3, 3), padding='same'), BatchNormalization(), Activation("relu"),
Conv2D(64, (3, 3), padding='same'), BatchNormalization(), Activation("relu"),
MaxPooling2D((2, 2)),
Dropout(0.1),

# 2nd Convolutional Block
Conv2D(128, (3, 3), padding='same'), BatchNormalization(), Activation("relu"),
Conv2D(128, (3, 3), padding='same'), BatchNormalization(), Activation("relu"),
MaxPooling2D((2, 2)),
Dropout(0.2),

# 3rd Convolutional Block
Conv2D(256, (3, 3), padding='same'), BatchNormalization(), Activation("relu"),
Conv2D(128, (3, 3), padding='same'), BatchNormalization(), Activation("relu"),
MaxPooling2D((2, 2)),
Dropout(0.4),

# Dense Layers
Flatten(),
Dense(128, kernel_regularizer=l2(0.005)), BatchNormalization(), Activation("relu"),
Dropout(0.3),

# Output Layer
Dense(10, activation='softmax')
])
```

Layer (type)	Output Shape	Param #
random_translation (RandomTranslation)	(None, 32, 32, 3)	0
random_flip (RandomFlip)	(None, 32, 32, 3)	0
conv2d (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256
activation (Activation)	(None, 32, 32, 64)	0
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 64)	256
activation_1 (Activation)	(None, 32, 32, 64)	0
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 128)	512
activation_2 (Activation)	(None, 16, 16, 128)	0
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 128)	512
activation_3 (Activation)	(None, 16, 16, 128)	0
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_1 (Dropout)	(None, 8, 8, 128)	0
conv2d_4 (Conv2D)	(None, 8, 8, 256)	295,168
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 256)	1,024
activation_4 (Activation)	(None, 8, 8, 256)	0
conv2d_5 (Conv2D)	(None, 8, 8, 128)	295,040
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 128)	512
activation_5 (Activation)	(None, 8, 8, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 128)	262,272
batch_normalization_6 (BatchNormalization)	(None, 128)	512
activation_6 (Activation)	(None, 128)	0
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

6.- OPTIMIZATION DETAILS

1. **RESEARCH:** Evaluate CNN models for image classification.
2. **START:** From a simple model with incremental steps.
3. **RECORD:** Every step taken and share with the other members .
4. **CHOSEN ONE:** Pick up the best model with the best results.
5. **RINSE AND REPEAT:** implementation of further optimization over best model.



TECHNIQUES IMPLMENTED:

- **CNN Architecture.**
- **Data Augmentation.**
- **Regularization.**
- **Optimizer Selection.**
- **Learning Rate Schedules**
- **Early stopping**
- **Hyperparameters tunning**
- **Epoch tunning**

7.- TRANSFER LEARNING

Models used for transfer learning

1. **ResNet50:** Bad accuracy before tunning. Achieved an accuracy score of 0.94 but with sever overfitting.

2. **VGG16:** Initially better than ResNet with 0.7 accuracy but unable to improve the model over ours.

3. **SqueezeNet:** Started the best with 0.8 in accuracy and val accuracy of 0.7 but unable to improve much further.

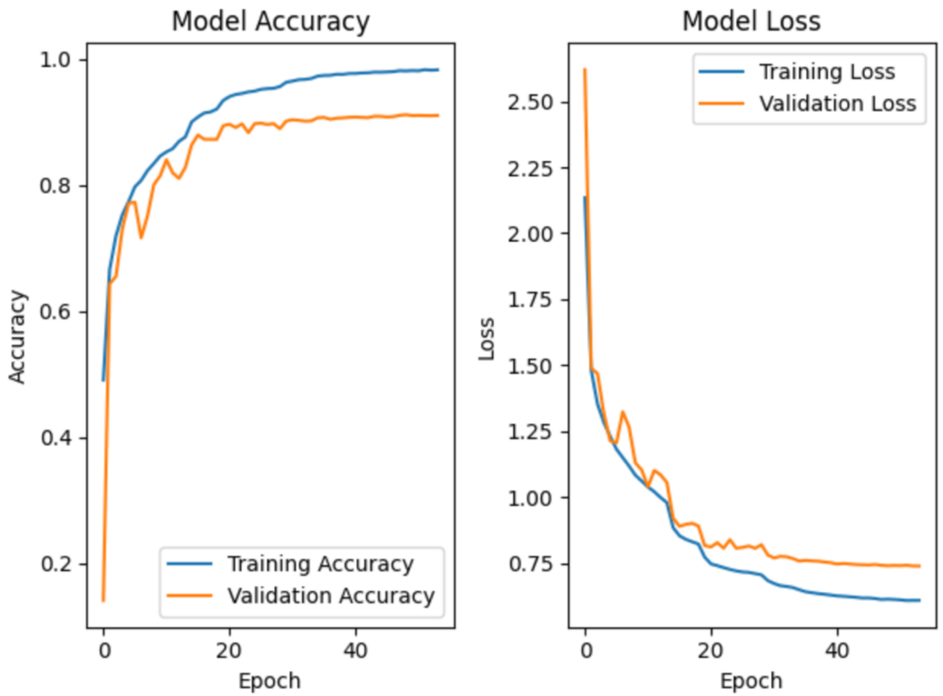
Conclusion

Unfreezing layers and tuning hyperparameters improved performance but revealed generalization limitations, especially in ResNet50. SqueezeNet showed stable learning, yet our custom model consistently achieved 0.9 accuracy, outperforming transfer learning models.

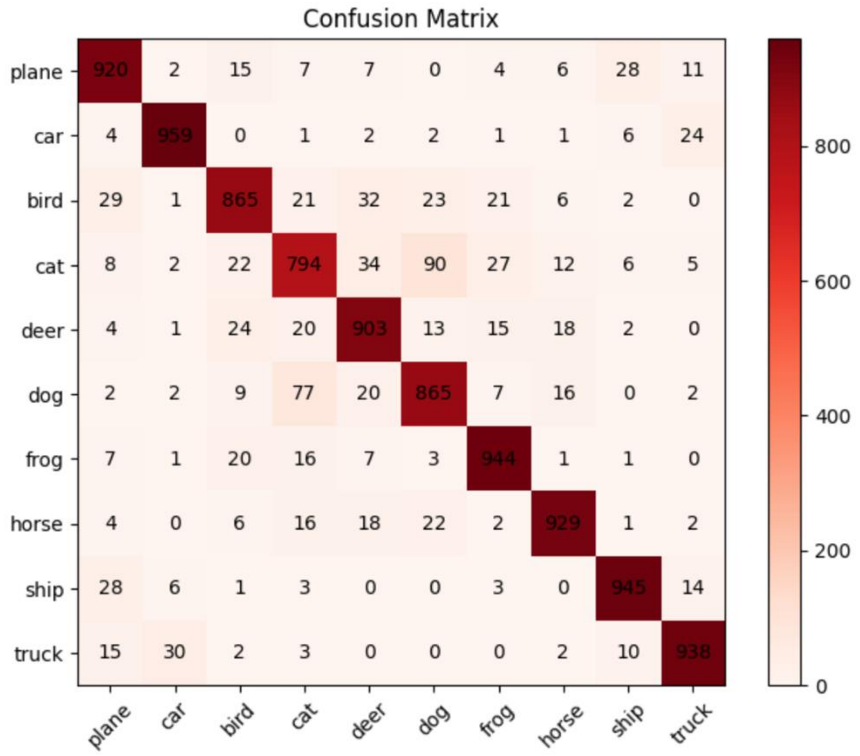
8.- EVALUATION

The main metrics we used to evaluate every version of the model were the **model loss and model accuracy** (cross entropy and loss and classification accuracy), **the confusion matrix, the classification report** and to make sure everything was working we plotted a **10 x 10 matrix with predictions** for our models than ran several times every iteration to see if our results were correct.

Here are shown all the evaluation metrics for our best working model.



Model Loss and accuracy



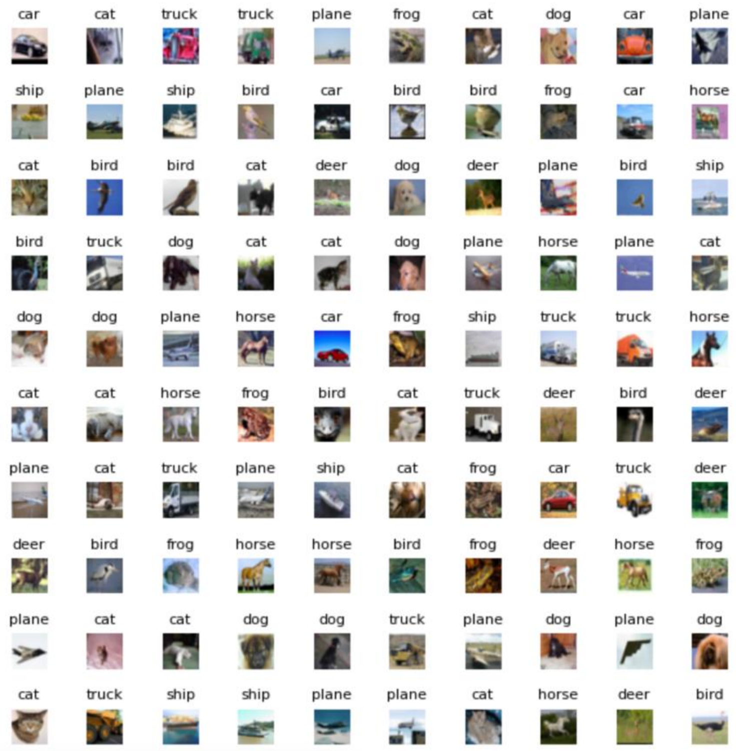
Confusion Matrix

8.- EVALUATION

Classification report

	precision	recall	f1-score	support
plane	0.90	0.92	0.91	1000
car	0.96	0.96	0.96	1000
bird	0.90	0.86	0.88	1000
cat	0.83	0.79	0.81	1000
deer	0.88	0.90	0.89	1000
dog	0.85	0.86	0.86	1000
frog	0.92	0.94	0.93	1000
horse	0.94	0.93	0.93	1000
ship	0.94	0.94	0.94	1000
truck	0.94	0.94	0.94	1000
accuracy			0.91	10000
macro avg	0.91	0.91	0.91	10000
weighted avg	0.91	0.91	0.91	10000

10 x 10 prediction matrix



100% Accuracy on this sample.

8.- EVALUATION

We even implemented a widget with ipywidgets with slider to make quick evaluations of our models and was also funny to use 😊

9.- MODEL DEPLOYMENT

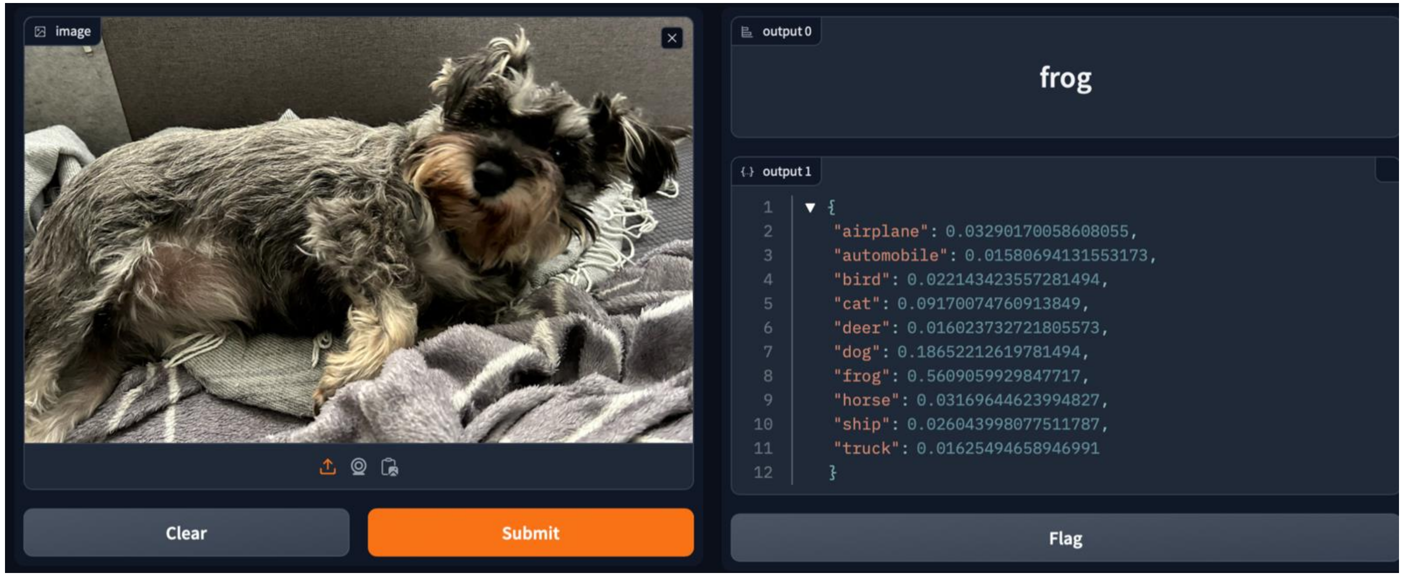
Started model deployment with Flask and tensorflow serving. We created the html file, we saved the model in keras, we created the folder architecture for the website, we installed Docker. And we almost make it run locally but there were many problems and compatibility issues in bash between tensorflow, docker and Pyhthon versions.

After creating many virtual environments and becoming a little bit crazy we changed to gradio and everything started to work.
Thanks Isabella.

Let us introduce you…

our model:

https://d047ff7ba1c11af4d5.gradio.live



A frog?

10.- WRAP UP

1. In this project, we explored image classification using CNNs on the CIFAR-10 dataset, starting from simple models and advancing through multiple optimizations and transfer learning techniques.
2. Despite experimenting with models like ResNet50, VGG16, and SqueezeNet, none were able to outperform our custom CNN, which consistently achieved an accuracy of 0.9.
3. Data augmentation, regularization, and careful hyperparameter tuning were key to preventing overfitting and achieving high performance.
4. Ultimately, we deployed our model using Gradio after encountering compatibility issues with TensorFlow Serving. The custom model remains our best-performing solution.
5. And las t but but not least: doing all this has been fun, entertaining and overall we learnt (just like in Sesame St).

10.- WRAP UP

Thank you !