CLASIFICADOR DE PLANTAS DE INTERIOR Y EXTERIOR TÓXICAS PARA FELINOS

INTELIGENCIA ARTIFICIAL I - E2

Por: Angélica María Sanabria Flórez - 2205564, Anyi Lorena Villabona Roa - 2215163 y Karen Juliana Mora Jaimes - 2202027

Universidad Industrial de Santander

DATASET

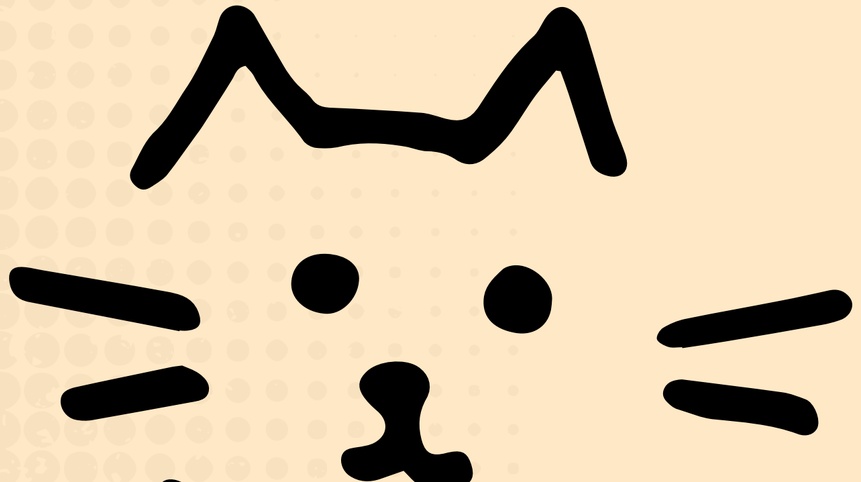
House Plant Species: Este conjunto consta de 8.250 imágenes categorizadas en 47 clases de especies de plantas de interior y exterior en el hogar recopiladas de Bing Images y seleccionadas manualmente por el autor de este dataset.

Composición del conjunto de datos:

- El número de imágenes por clase se distribuye en 4.125 especies tóxicas y 4.125 especies no tóxicas

Características de la imagen:

- Las imágenes varían en calidad y resolución.
- El conjunto de datos incluye imágenes de plantas completas y primeros planos de partes específicas de la planta.
- Las plantas se colocan en interiores y exteriores.
- Las imágenes están organizadas en carpetas separadas para cada categoría de planta.



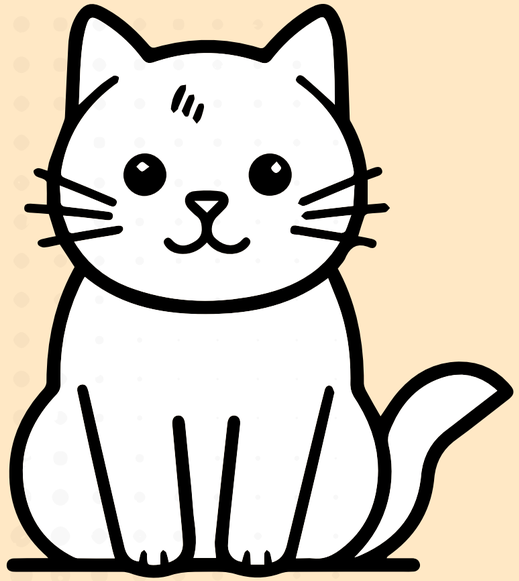
INTRODUCCIÓN

El contacto entre los felinos y las plantas que rodean su alrededor son un factor muy importante para su bienestar, principalmente cuando se trata de especies que son potencialmente tóxicas para ellos. Es por esto, que surge la necesidad de identificar con precisión cuáles especies son tóxicas y cuáles no.

Con este propósito, el siguiente proyecto consistió en desarrollar un clasificador donde se implementaron técnicas avanzadas de aprendizaje automático y redes neuronales que incluyen estimadores como DecisionTree, RandomForest y SupportVectorMachine, junto con modelos basados en perceptrones multicapa con tres, seis y diez capas ocultas.

OBJETIVO

Desarrollar un sistema de clasificación que identifique plantas de interior como tóxicas o no tóxicas para los felinos, con el fin de promover la seguridad de las mascotas y mejorar la toma de decisiones de los dueños al elegir plantas para sus hogares.



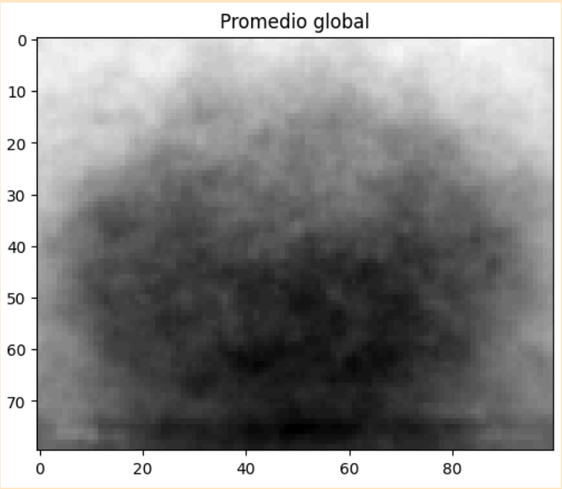
PRIMERA ENTREGA

TAREA:

1 Primera parte

Calcular el promedio de todas las imágenes pertenecientes a la especie African Violet (Saintpaulia ionantha) y graficarlo con el comando plt.imshow().

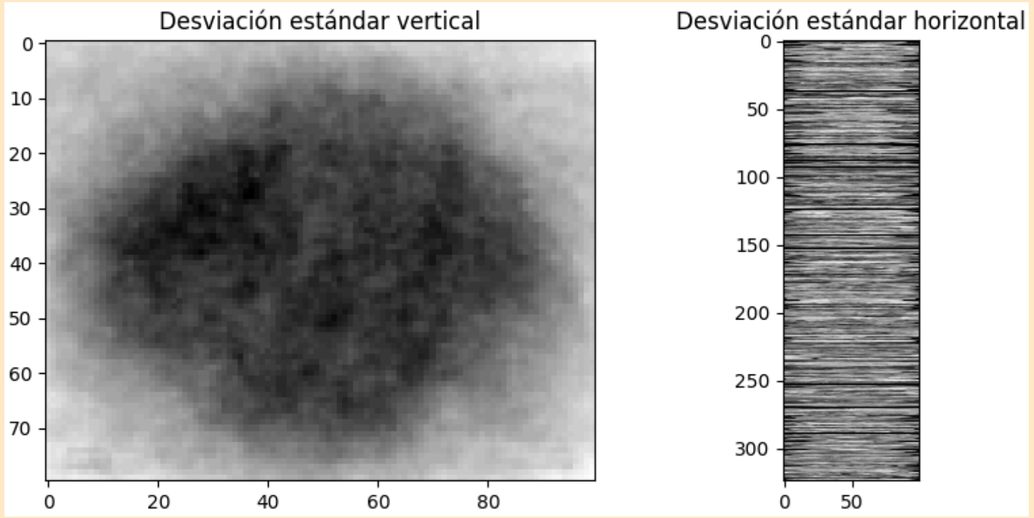
```
b = "/content/drive/MyDrive/IA-1/especies-no-toxicas/African Violet (Saintpaulia ionantha)"
mi_lista=os.listdir(os.path.join(b))
total=np.zeros((80,100))
n=len(mi_lista)
for i in mi_lista:
  img = mpimg.imread(os.path.join(b,i))
  img=img[:,:,0]
  total=total+img
avg_b=total/n
plt.imshow(avg_b, cmap= "gray")
plt.title("Promedio global")
plt.show()
```

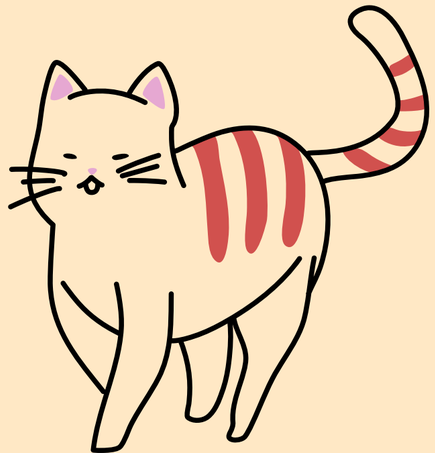


2 Segunda parte:

Calcular la desviación estándar de todas las imágenes pertenecientes a la especie African Violet (Saintpaulia ionantha) y graficarlo con el comando plt.imshow().

```python
b = "/content/drive/MyDrive/IA-1/especies-no-toxicas/African Violet (Saintpaulia ionantha)"
all_img = []
mi_lista = os.listdir(os.path.join(b))
for i in mi_lista:
  img = mpimg.imread(os.path.join(b,i))
  img = img[:,:,0]
  all_img.append(img)
#Desviación estándar para el eje y
std_v = np.std(np.array(all_img), axis=0)
#Desviación estándar para el eje x
std_h = np.std(np.array(all_img), axis=1)
#Creación de subplots
fig, axes = plt.subplots(1, 2, figsize=(9, 4))
#Ploteo de ambos subplots
axes[0].imshow(std_v, cmap="gray")
axes[0].set_title("Desviación estándar vertical")
axes[1].imshow(std_h, cmap = "gray")
axes[1].set_title("Desviación estándar horizontal")
#Ajustar espaciado
plt.tight_layout()
plt.show()
```





SEGUNDA ENTREGA

TAREA:

1 Primera parte

Hacer uso de **TODOS** los estimadores vistos en clase: DecisionTree, RandomForest, SupportVectorMachine, con PARÁMETROS POR DEFECTO para clasificación.

DecisionTreeClassifier

```python
from sklearn.tree import DecisionTreeClassifier

est_DT = DecisionTreeClassifier()
est_DT.fit(X_train_f, y_train)
y_pred = est_DT.predict(X_test_f)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy del DecisionTreeClassifier: {accuracy:.4f}")

```

Accuracy:

0.5702

RandomForestClassifier

```python
from sklearn.ensemble import RandomForestClassifier

est_RF = RandomForestClassifier()
est_RF.fit(X_train_f, y_train)
y_pred = est_RF.predict(X_test_f)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy del RandomForestClassifier: {accuracy:.4f}")

```

Accuracy:

0.6422

SupportVectorClassifier

```python
from sklearn.svm import SVC

est_SVC = SVC()
est_SVC.fit(X_train_f,y_train)
y_pred = est_SVC.predict(X_test_f)
accuracy = accuracy_score(y_test, y_pred)

```

Accuracy:

0.6704

2 Segunda parte:

a) usando train_test_split:

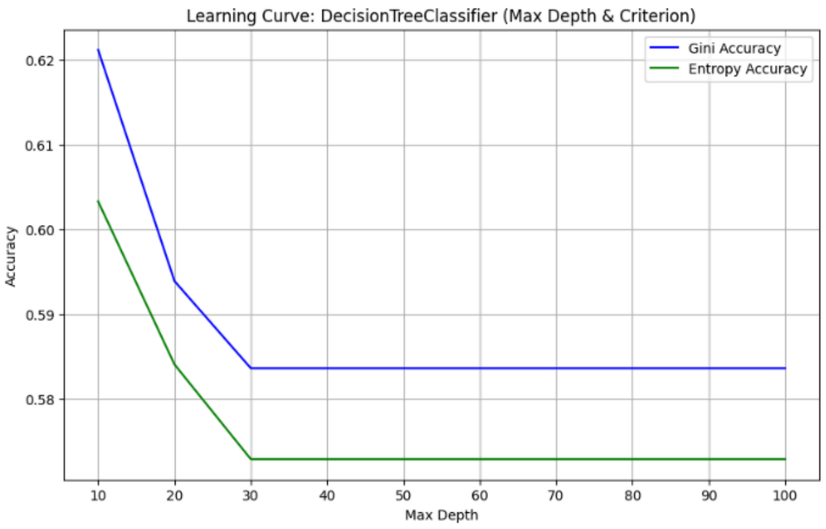
Ejecute el tuning de parámetros (learning curves) del notebook 11

- DecisionTree (max_depth, criterion)
- RandomForest (n_estimators, criterion)
- SVC (kernel, gamma)

b) cross_val_score:

- DecisionTree
- RandomForest
- SVC

DecisionTreeClassifier (train_test_split)

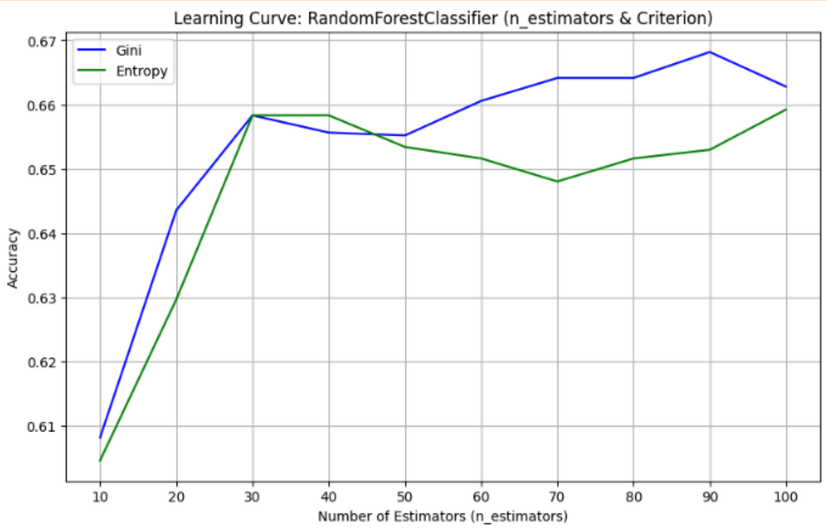


La gráfica muestra la curva de aprendizaje para un modelo de DecisionTreeClassifier, comparando las métricas de accuracy para criterion Gini y Entropy con respecto a max depth.

La curva de aprendizaje Gini comienza más alta que la curva de Entropy, pero se van convergiendo a medida que aumenta el max_depth, lo que sugiere que ambos criterion tienen un desempeño similar para profundidades mayores.

La curva parece estabilizarse alrededor de un max_depth de 30, lo que indica que aumentar el max_depth más allá de ese punto no mejora significativamente la precisión del modelo.

RandomForestClassifier (train_test_split)



En este caso, ambas curvas de aprendizaje aproximadamente inician en valores alrededor del 0.61 de accuracy y luego aumentan progresivamente al incrementar el número de estimadores. La curva se estabiliza alrededor de los 70-80 árboles.

En general, la gráfica muestra que el modelo de RandomForest tiene un buen desempeño alcanzando un accuracy del 66% con un número relativamente pequeño de árboles, siendo mejor que el DesicionTree que obtuvo 62%.

SupportVectorClassifier (train_test_split)

```python
def show_curve_svc(X_train_f, X_test_f, y_train, y_test):
    # Lista de kernels a evaluar
    kernels = ['linear', 'rbf', 'poly', 'sigmoid']
    # Valores de gamma a evaluar
    gamma_range = np.logspace(-3, 2, 6)

    # Inicializar diccionario para guardar los resultados
    results = {kernel: [] for kernel in kernels}

    for kernel in kernels:
        for gamma in gamma_range:
            # Crear y entrenar el modelo con los parámetros actuales
            model = SVC(kernel=kernel, gamma=gamma, random_state=42)
            model.fit(X_train_f, y_train)
            # Predecir y calcular la precisión
            pred = model.predict(X_test_f)
            accuracy = accuracy_score(y_test, pred)
            results[kernel].append(accuracy)
```

Para el SVC hubo bastantes limitaciones, debido a que no pudo graficarse la curva de aprendizaje principalmente por la complejidad y los requisitos computacionales del mismo lo que provocó problemas de conectividad en el entorno de trabajo de Colab, en donde aumentó significativamente el tiempo de entrenamiento y la carga computacional, al igual que la gran cantidad de datos con la que trabajamos.

Posibles soluciones:

- Reducir el tamaño de datos
- Utilizar un entorno de ejecución diferente

DecisionTreeClassifier (cross_val)

```
#@title **code** using DTC
from sklearn.tree import DecisionTreeClassifier
show_curve(DecisionTreeClassifier())
```

RandomForestClassifier (cross_val)

```
#@title **code** using RFC
from sklearn.ensemble import RandomForestClassifier
show_curve(RandomForestClassifier())
```

SupportVectorClassifier (cross_val)

```
#@title **code** using SVC
from sklearn.ensemble import SVC
show_curve(SVC())
```

Para el desarrollo de la parte b) que consistió en usar cross-validation, nuevamente se presentaron limitaciones de recursos computacionales similares a lo sucedido con el estimador SVC con train_test_split. El conjunto de datos y la complejidad de los modelos requerían de más memoria, CPU y GPU de lo que Colab podía proporcionar.

Por otra parte, el tiempo necesario para completar el proceso de cross-validation era demasiado prolongado, lo que provocaba problemas de estabilidad y desconexión del entorno de ejecución.

TERCERA ENTREGA

TAREA:

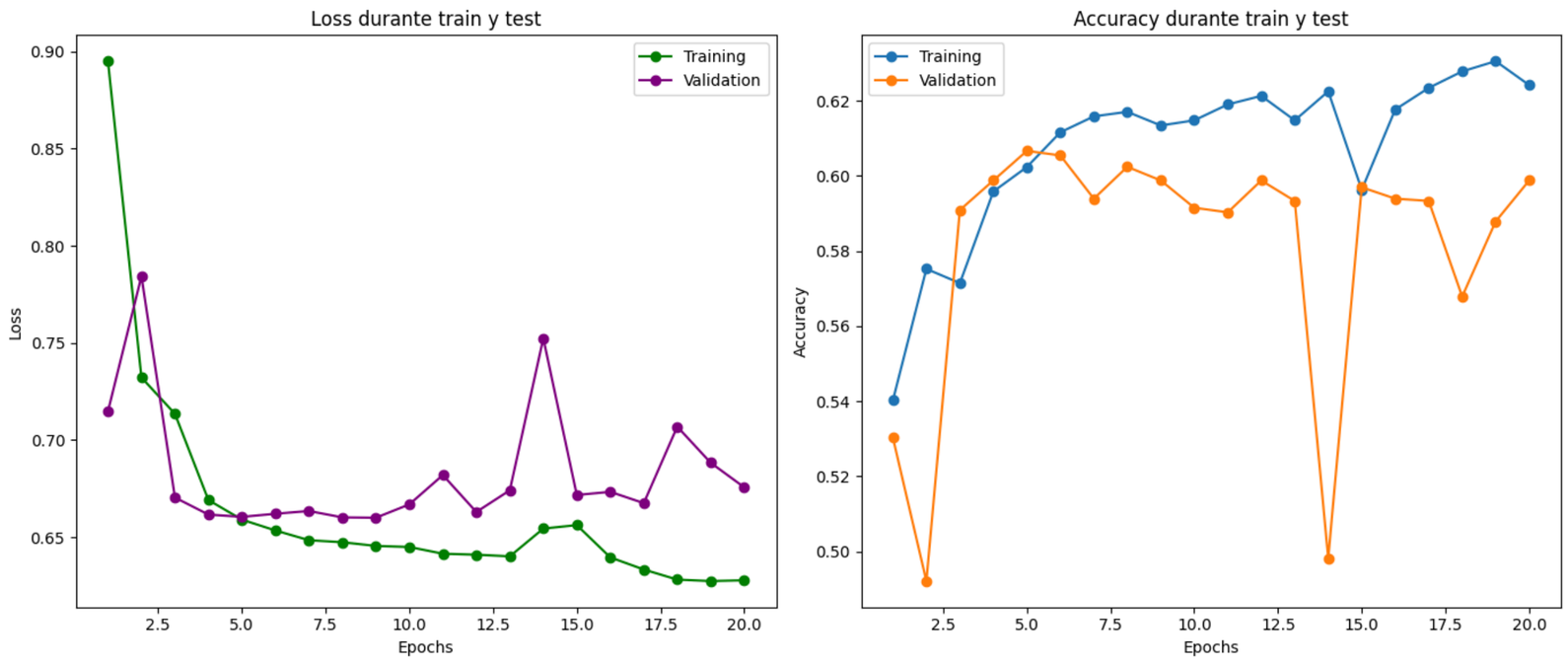
Agregar la implementación de un perceptrón multicapa (TAL COMO SE VIÓ EN LOS NOTEBOOKS 11 y 12) con: 3 capas ocultas, 6 capas ocultas y 10 capas ocultas. Calcular el accuracy_score para cada uno de estos.

Perceptrón multicapa con tres capas ocultas

Número de parámetros train: **3,076,801**

Número de parámetros test: **3,076,801**

Accuracy: **0.5987**

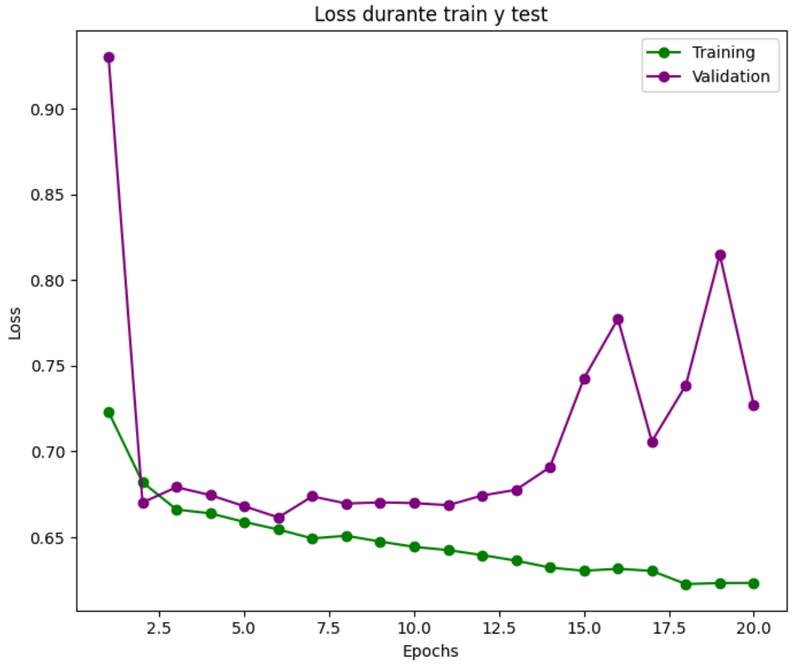


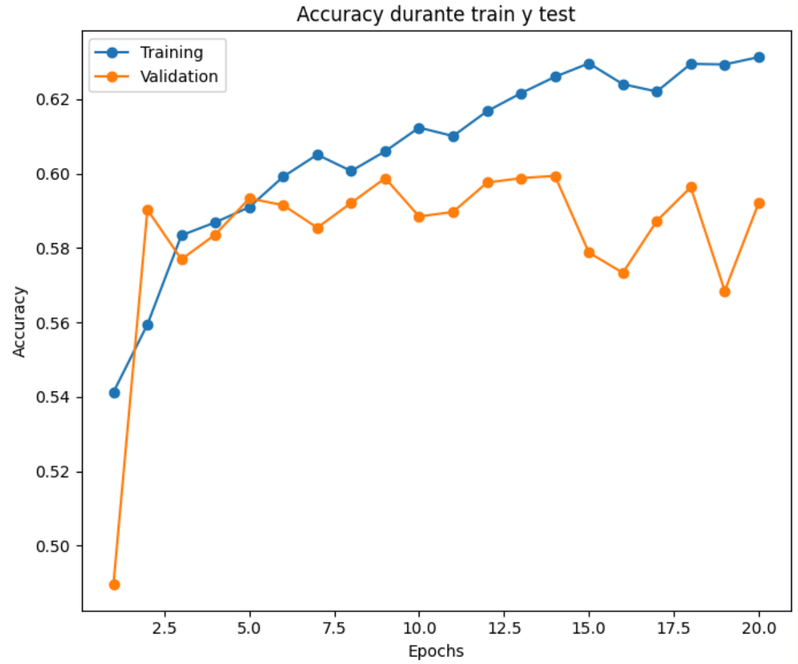
Perceptrón multicapa con seis capas ocultas

Número de parámetros train: **3,107,329**

Número de parámetros test: **3,107,329**

Accuracy: **0.5921**



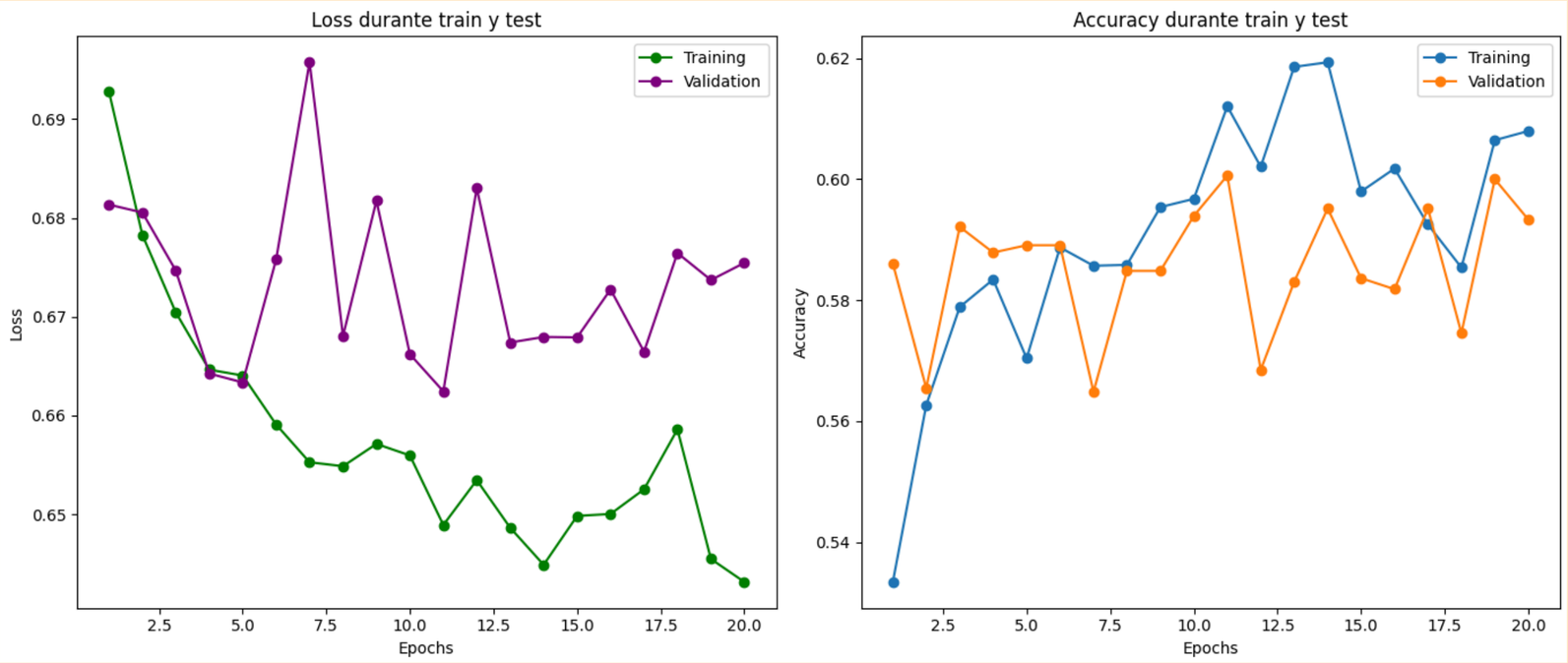


Perceptrón multicapa con diez capas ocultas

Número de parámetros train: **3,159,425**

Número de parámetros test: **3,159,425**

Accuracy: **0.5933**



CONCLUSIÓN

En conclusión el desarrollo del proyecto demostró la posibilidad de clasificar imágenes de plantas tóxicas y no tóxicas para felinos, utilizando estimadores tradicionales y redes neuronales. Si bien los resultados actuales muestran un desempeño moderado, existe un margen de mejora considerable mediante la optimización de hiperparámetros y la implementación de regularización. En general, el modelo SVC y el perceptrón multicapa de 3 capas ocultas destacaron como las configuraciones más óptimas dentro de las limitaciones computacionales actuales.

Además, se identificaron varias áreas de mejora que incluyen la regularización mediante la implementación de dropout o penalizaciones como L1/L2 en los modelos de redes neuronales, testeo de modelos especializados como redes convolucionales para mejorar la extracción de las características visuales y la exploración de entornos con mayor capacidad de procesamiento para manejar modelos más complejos y cross-validation.

¡GRACIAS!