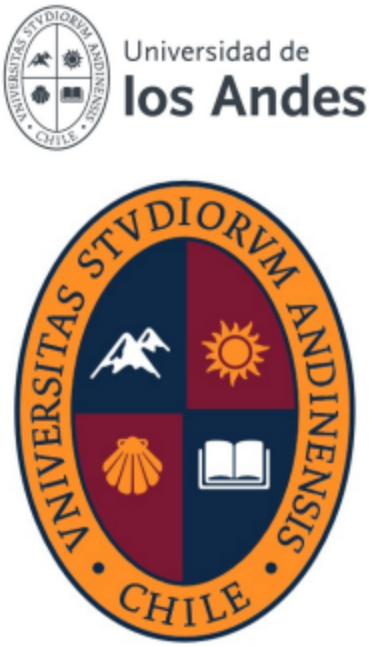


Universidad de los Andes

Facultad de Ingeniería y Ciencias Aplicadas.

Artificial Intelligence

Informe 4:

Evaluación de Encoders por medio de MLP, SVM y RF.

Diego Eyzaguirre

Eduardo Saavedra

Santiago, Chile

Junio, 2025

Abstracto

En el informe a continuación se evalúa el desempeño de los encoders DINOv2, CLIP y ResNet por medio del análisis de los resultados obtenidos a partir de la clasificación realizada sobre el espacio de características generado por estos encoders, utilizando MLP, SVM y Random Forest como modelos clasificadores. Adicionalmente se evalúa el desempeño de estos modelos de clasificación al trabajar con diferentes encoders, para así poder encontrar la mejor combinación de encoder y clasificador entre los mencionados.

Para poder comparar los resultados se extrajo el accuracy de cada encoder con cada clasificador y se comparó, obteniendo como resultado que DINOv2 es el mejor encoder, seguido muy de cerca por CLIP y por último ResNet. También se compararon los diferentes clasificadores entre sí, y se descubrió que SVM resultó ser el más parejo y con mejor desempeño al trabajar con cada encoder, con un accuracy promedio de 84.66%. Eso sí, la MLP fue el modelo clasificador que obtuvo el resultado más alto al trabajar con DINOv2 con una accuracy de 94.78%, mientras que Random Forest resultó ser aproximadamente un 10% peor en promedio que la SVM.

Estos resultados nos permiten concluir que el mejor encoder es DINOv2, pero cuando se trata de seleccionar el mejor modelo clasificador, este podría ser tanto SVM como MLP, pues SVM tiene un desempeño promedio más alto, pero la MLP tiene un desempeño puntual mayor con algunos encoders, por lo que el modelo clasificador a utilizar va a depender del encoder que se desee utilizar y de la complejidad del trabajo de clasificación asociado. También se concluyó que Random Forest podría ser utilizado en casos en los que se desee tener una respuesta interpretable por parte del modelo, aunque su desempeño sea menor y lo haga menos preferible para casos de aplicación real.

Index

1. Introducción

En el mundo de machine learning, existen encoders muy sofisticados capaces de extraer la mayoría de las características relevantes de una entrada, como sucede con DINOv2, CLIP y ResNet. Una manera de utilizar lo que entregan estos modelos es tomar la salida y clasificarla, lo cual se puede hacer de manera muy simple utilizando una MLP, SVM or Random Forest, pues la información ya fué procesada por el encoder y la tarea de clasificar esto es liviana. Utilizando estos algoritmos clasificadores, es posible obtener diferentes formas de clasificación, algunas más eficientes que otras, en base a el espacio de características con el que se está trabajando.

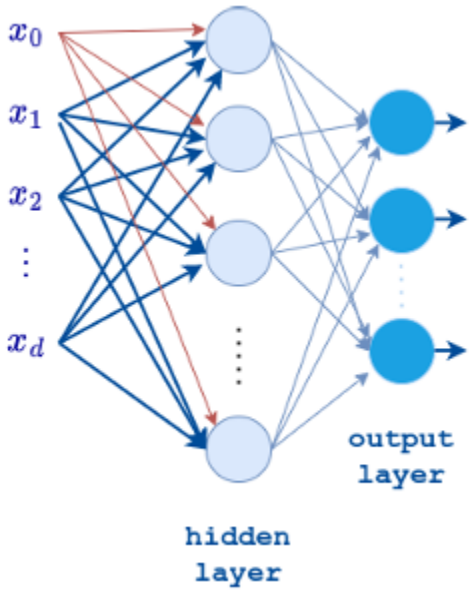
El uso de estos modelos de clasificación simples reduce mucho el costo de producción, pues la mayoría de las tareas relacionadas a clasificación en machine learning se ven reducidas a encontrar un encoder pre-entrenado con el tipo de datos que deseamos evaluar, y acoplar este encoder a uno de los modelos de clasificación mencionados anteriormente, los cuales no requieren muchos datos para ser entrenados y son capaces de clasificar con una precisión acorde a la capacidad del encoder en cuestión.

Esta tarea busca utilizar los modelos de clasificación para evaluar el desempeño de distintos encoders, para así poder determinar cuál se ajusta mejor a la tarea de reconocimiento de imágenes por clases, y además se busca determinar cual modelo de clasificación entrega los mejores resultados, pues esto nos indicaría de qué manera está estructurado el espacio de características generado por lo encoders.

1.1. MLP

Un Multi-Layer Perceptron (MLP) es una extensión del modelo computacional de una neurona biológica, conocido como perceptrón. El comportamiento de un solo perceptrón, o neurona, se asemeja al de un modelo de regresión logística. Al combinar y conectar múltiples perceptrones, los MLP forman un tipo de red neuronal organizada en capas, donde la salida de una neurona se convierte en la entrada de otra. Una red en la que cada neurona está conectada con todas las neuronas de la capa adyacente se conoce como “fully connected”.

Figura 1.2: Multi Layer Perceptron



Fuente: Fundamentals of Machine Learning (José Manuel Saavedra)

Las neuronas en las capas internas, también llamadas capas ocultas, utilizan funciones no lineales y diferenciales para transformar las entradas capa por capa, facilitando que la salida final sea más fácil de clasificar.

1.2. Distancias Espaciales

La mayoría de las entradas a un modelo de aprendizaje automático pueden representarse como puntos en un espacio multidimensional. En este espacio, la distancia entre dos puntos sirve como una medida de su similitud: los puntos más cercanos son más parecidos, mientras que los más lejanos son menos similares. Sin embargo, dado que el espacio es multidimensional, el método utilizado para calcular esa distancia puede variar, y cada método capta una noción distinta de similitud.

a. Distancia Coseno

La distancia coseno mide el ángulo entre dos vectores que se originan en el origen y apuntan hacia los puntos de entrada. Se enfoca en la orientación más que en la magnitud de los vectores. Una similitud coseno de 1 (o un ángulo de 0°) significa que los vectores apuntan en la

misma dirección (máxima similitud), mientras que una similitud de -1 (o 180°) indica que apuntan en direcciones opuestas (máxima disimilitud).

$$cos(\theta) = \frac{A \cdot B}{||A|| \cdot ||B|}$$

b. Distancia Euclideana

La distancia euclidiana representa la diagonal entre los dos puntos a comparar; matemáticamente, es como trazar una línea recta de uno al otro, y esa distancia indica qué tan similares son (cuanto menor sea la distancia, mayor la similitud).

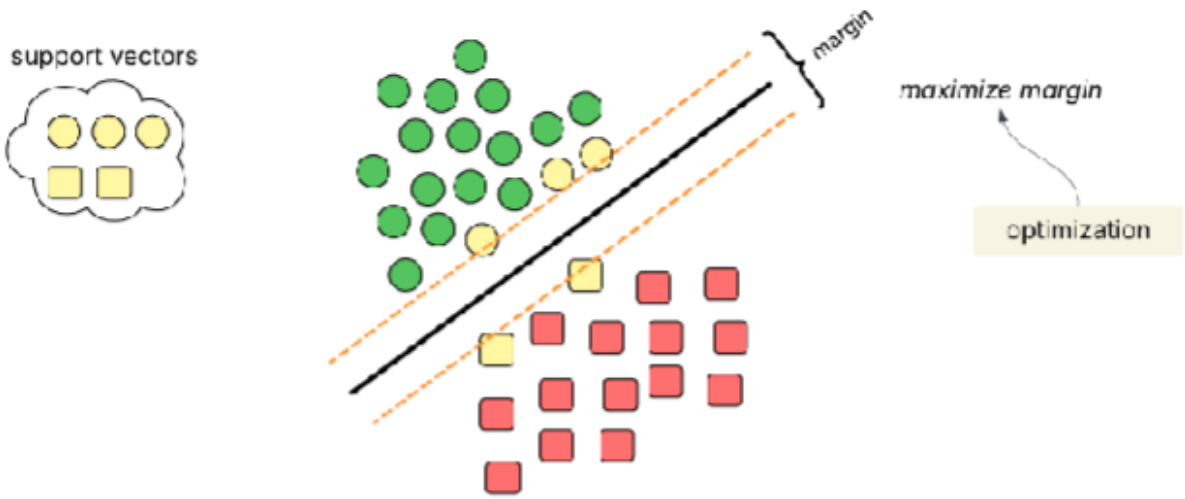
$$distance(a, b) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

1.3. SVM

Las máquinas de soporte de vectores (SVM) son un modelo simple de clasificación binaria basado en la estimación de un hiperplano que representa la separación espacial óptima de las clases en el espacio. En base a este hiperplano se puede inferir y clasificar un dato como perteneciente a una clase u a otra dependiendo si se encuentra por encima o bajo el hiperplano.

Para encontrar este hiperplano se debe entrenar el modelo para encontrar el más óptimo en relación a los datos de entrenamiento, y para lograr esto, se mide la distancia perpendicular entre un punto de una de las dos clases y el hiperplano. Estos puntos van a ser los más cercanos al hiperplano y se les llama vectores de soporte, pues el hiperplano se posiciona en una distancia tal que maximice la distancia entre los vectores de soporte de ambas clases. Adicionalmente, el modelo toma como hiper-parámetro un margen que tiene la funcionalidad de definir el la distancia mínima que deben de tener los vectores de soporte al hiperplano, y mientras menor es este margen, mayor es el bias del modelo pues es más estricto al momento de clasificar.

Figura 1.7: Representación de una SVM



Fuente: Fundamentals of Machine Learning (José Manuel Saavedra)

Es posible crear un clasificador multiclases con SVM, pero es necesario crear varios hiperplanos binarios con el método *"one-vs-all"* que consiste en realizar una comparación entre una clase contra todo el resto para crear cada hiperplano, resultando en tantos hiperplanos como clases existentes.

1.4. Árboles de Decisión

Un árbol de decisión es un modelo jerárquico para aprendizaje supervisado en donde el espacio es dividido en regiones que discriminan entre clases, y estas regiones se generan por medio de una secuencia sucesiva de divisiones (splits), siguiendo una estrategia similar a divide y conquista. Cada división representa un nodo y los nodos terminales, en donde se genera la predicción de clase, se llaman nodos hoja.

Los nodos en sí pueden ser univariados (el split se hace con respecto a una variable) o multivariados (el split es con respecto a 2 o más variables). Los nodos hoja pueden representar una clase o también en casos particulares pueden ser una regresión numérica, entregando un

valor o probabilidad para el input dado. Los nodos hoja también pueden no ser únicos, dicho de otra forma, puede haber más de un nodo hoja para la misma clase, y los splits pueden dividir el espacio en más de dos partes.

Para realizar los splits en la etapa de entrenamiento, se utiliza un algoritmo *greedy* que nos permite determinar cual split es el más conveniente para nuestro espacio. Esto se calcula por medio de la entropía del sistema, lo ideal es conseguir un split que reduzca la entropía de ambos grupos generados al mínimo, lo cual representaría que ambos grupos contienen únicamente datos de la misma clase. En el caso de no alcanzar la entropía mínima (0), es necesario mantener el split que genere la mayor cantidad de entropía con respecto al espacio original, y este paso se repite hasta lograr grupos indivisibles.

1.5. Random Forest

Random Forest es un modelo de aprendizaje supervisado basado en un conjunto de árboles de decisión, diseñado para mejorar la capacidad predictiva y reducir el sobreajuste mediante la combinación de múltiples modelos individuales. Esta técnica se enmarca dentro de los métodos de *ensemble* y sigue el principio de "sabiduría de la multitud": al combinar varios modelos débiles (en este caso, árboles), se obtiene un modelo más robusto y preciso.

Cada árbol dentro de un Random Forest se entrena de manera independiente siguiendo dos estrategias principales que introducen aleatoriedad al sistema:

1. Bootstrap Aggregation (Bagging): Cada árbol se entrena con una muestra aleatoria (con reemplazo) del conjunto de datos original. Esto genera una diversidad entre los árboles al cambiar ligeramente la distribución de los datos de entrenamiento.

2. Selección aleatoria de variables: En cada nodo donde se debe realizar un split, en lugar de considerar todas las variables disponibles, se selecciona un subconjunto aleatorio de variables. Esto evita que todos los árboles aprendan patrones similares y ayuda a disminuir la correlación entre ellos.

El proceso de decisión en un Random Forest es colectivo: una vez entrenados los árboles, la predicción se realiza mediante un voto mayoritario en el caso de clasificación (es decir, la clase que más árboles predigan), o mediante el promedio de predicciones en el caso de regresión.

Esta combinación de múltiples árboles permite que el modelo sea más resistente al sobreajuste en comparación con un solo árbol de decisión. A su vez, al incorporar aleatoriedad tanto en los datos como en las variables, se fomenta la diversidad entre los modelos, lo cual es clave para mejorar el rendimiento general del sistema.

Si bien cada árbol individual sigue un enfoque similar al del árbol de decisión clásico, usando divisiones jerárquicas basadas en métricas como entropía, la agregación en Random Forest permite compensar errores individuales y generalizar mejor a datos no vistos.

1.6. Encoder

Un encoder es un componente de una red neuronal diseñado para transformar datos de entrada (como imágenes, texto o audio) en una representación compacta e informativa, normalmente un vector de tamaño fijo o un mapa de características de menor dimensión. Esta representación captura las características más relevantes de la entrada, descartando la información redundante o irrelevante. Los codificadores se usan comúnmente como la primera etapa en muchas arquitecturas de machine learning y deep learning, y por lo general se reutilizan; sólo se modifica el decodificador para clasificar en distintas categorías.
El rol principal del codificador es extraer patrones o características significativas a partir de los datos crudos de entrada, permitiendo que el resto del modelo realice tareas como clasificación, traducción o generación. Dependiendo del tipo de entrada, se pueden usar distintas arquitecturas como codificadores.
Al aprender representaciones compactas, los codificadores ayudan a reducir la dimensionalidad, mejorar la eficiencia del modelo y facilitar la generalización en diferentes tareas.

1.7. Vision Transformer

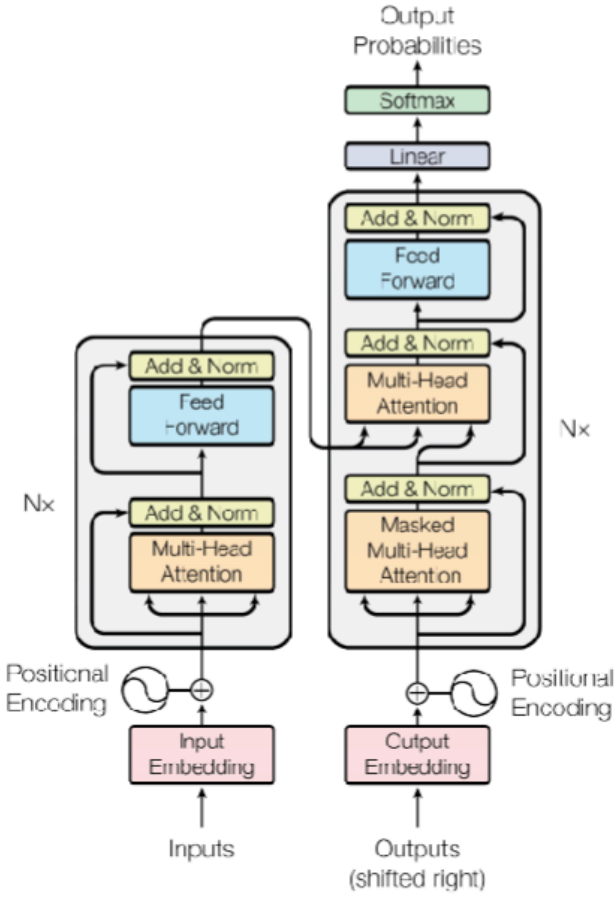
El Vision Transformer (ViT) es una arquitectura de modelo que procesa imágenes usando la misma idea que los transformadores en modelos de lenguaje. En lugar de usar filtros como en

las CNNs, ViT divide una imagen en pequeños parches y los trata como si fueran una secuencia de palabras.

Cada parche de imagen se convierte en un vector (embedding), y se añade un token especial que representa toda la imagen. Luego, el modelo agrega información sobre la posición de cada parche y pasa todo por capas de transformador, que usan mecanismos de atención para encontrar relaciones entre los parches.

Al final, el modelo utiliza ese token especial para interpretar la imagen completa y hacer predicciones. ViT funciona especialmente bien cuando se entrena con grandes conjuntos de datos de imágenes y se usa ampliamente como codificador en muchos modelos de visión modernos.

Figura 1.8: Arquitectura ViT



Fuente: Fundamentals of Machine Learning (José Manuel Saavedra)

1.8. ResNet

ResNet, abreviatura de Residual Network (Red Residual), es una arquitectura que permite construir redes neuronales muy profundas manteniendo un entrenamiento eficaz. Lo logra mediante el uso de bloques residuales, que incluyen conexiones de salto (*skip connections*) que permiten al modelo omitir una o más capas. Estos atajos o *shortcuts* ayudan a mitigar el problema del desvanecimiento del gradiente, asegurando que las capas más profundas aún puedan aprender características significativas sin deteriorar el rendimiento del modelo. Existen distintas versiones de ResNet según la cantidad de capas, siendo las más comunes ResNet18 y ResNet32.

Al permitir que la información fluya directamente a través de la red, incluso a lo largo de muchas capas, ResNet hace posible que el modelo retenga y refine las características aprendidas en etapas anteriores. Esta arquitectura es especialmente adecuada para redes convolucionales profundas, donde supera de forma consistente a modelos tradicionales que no cuentan con conexiones residuales, incluso sin utilizar técnicas como la congelación o desactivación de capas.

Figura 1.9: Arquitectura de ResNet para ImageNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Fuente: Deep Residual Learning for Image Recognition

1.9. CLIP

El codificador CLIP (*Contrastive Language–Image Pre-training*) es una herramienta basada en redes neuronales diseñada para transformar imágenes en representaciones de características ricas y de alta dimensión. Actúa como la columna vertebral visual en la arquitectura de doble codificador de CLIP, la cual alinea datos visuales y textuales en un espacio de *embedding* compartido.

El codificador de imágenes de CLIP utiliza una arquitectura basada en un ResNet modificado o un *Vision Transformer* (ViT), pero en este caso trabajaremos con la versión basada en ViT. La principal diferencia entre CLIP y los codificadores mencionados anteriormente es que, durante el entrenamiento, CLIP aprende a proyectar las imágenes en un espacio de *embedding* en el que sus representaciones estén estrechamente alineadas con las descripciones textuales correspondientes. Esto se logra mediante un objetivo de aprendizaje contrastivo que maximiza la similitud entre pares de imagen-texto coincidentes, y la minimiza para los pares que no coinciden.

1.10. DINOv2

DINOv2 está construido sobre una arquitectura de Transformer Visual (ViT), funcionando como un potente encoder de imágenes en aprendizaje auto-supervisado. A diferencia de los encoders convolucionales tradicionales, DINOv2 se basa en la capacidad del transformer para modelar dependencias de largo alcance y contexto global mediante mecanismos de self-attention.

El codificador procesa las imágenes de entrada dividiéndolas primero en fragmentos de tamaño fijo, que luego son incrustados linealmente y combinados con encoders posicionales. Estas incrustaciones de fragmentos se pasan por múltiples capas de transformer, cada una

compuesta por mecanismos de multi-head self-attention y redes feedforward, permitiendo al modelo capturar representaciones jerárquicas y semánticas de la imagen.

Una característica clave del codificador de DINOv2 es el uso de un framework maestro-estudiante, donde el encoder estudiante es entrenado para igualar la salida de un encoder maestro que se actualiza lentamente. Esta arquitectura ayuda al modelo a aprender representaciones estables y generalizables. Los encoders comparten la misma columna vertebral ViT, pero operan bajo diferentes vistas aumentadas de la entrada, promoviendo la consistencia entre perspectivas variadas.

2. Metodología

2.1 Librerías

Las librerías utilizadas para realizar esta tarea fueron torch, torchvision, sklearn, joblib, clip, PIL, numpy, os, seaborn y matplotlib.

Desde torch se importan las herramientas útiles para implementar los modelos de ML y diseñar la MLP, como DataLoader y TensorDataset, la librería permite cargar el modelo DINOv2. Desde torchvision se carga el modelo ResNet34, de la librería se obtiene el módulo transforms, con el cual se definirán las transformaciones a aplicar a las imágenes del dataset antes de pasarlas por el modelo. Desde sklearn se obtiene la SVM a través de svc, y para RF mediante RandomForestClassifier, además se importan varias herramientas útiles como LabelEncoder, y métricas como accuracy_score y classification_report. La librería joblib se ocupa para guardar los modelos de RF, SVM y sus LabelEncoders. De clip se importa el encoder clip. PIL se utiliza para cargar y recortar las imágenes de entrada. Numpy se utiliza para hacer uso de arrays y por compatibilidad con demás librerías. Os se utiliza para el manejo de rutas hacia archivos, seaborn para obtener una confusion matrix, y matplotlib se usa para realizar el gráfico de barras de accuracy por clase.

2.2 Extractor de Características

Para evaluar los encoders a través de los distintos métodos de clasificación, primero se requiere de sus espacios de características, los features vectores que servirán de entrada. Para ello se elabora el código feature_extract.py, en cual se elige el encoder a usar entre DINOv2, ResNet34 y CLIP, cargando el correspondiente. Teniendo definido el dataset a usar como VocPascal, se elige además si usar las imágenes de entrenamiento o validación. Leyendo el archivo ‘train_voc.txt’ o ‘val_voc.txt’ se obtienen las labels de cada imagen, el nombre y las coordenadas que definen el bounding box del objeto de interés, esta información se guarda en las listas labels,files y box correspondientemente. Recorriendo la lista files, se construye la ruta hacia la imagen, y haciendo uso de PIL se carga y recorta, para luego ser preprocesada a través de una composición de transformaciones realizada con el módulo transforms de torchvision, esta secuencia de transformaciones es esperada por los modelos a usar y consisten en cambiar el tamaño, pasar tensor y normalizar. Hecho esto se cuenta con un tensor

al que se añade una dimensión de batch, y se pasa por el modelo elegido, cada predicción se agrega a un numpy array que es finalmente guardado a una ruta elegida junto a las labels del dataset, distinguiendo en el nombre si el espacio extraído es de validación o entrenamiento.

2.3 MLP

La clasificación a través de una MLP se lleva a cabo pasando el vector de entrada por una capa de 256 neuronas, y una capa final de clasificación con 20 neuronas para 20 clases. Esto se lleva a cabo en el código 'MLP.py', para construir el modelo se define la clase Simple_model, como subclase de nn.Module, clase base de las redes neuronales en torch. Se establecen las 2 capas necesarias como nn.Linear, y la función de activación como ReLU, así el forward queda definido por realizar flatten sobre la entrada, pasarla por la primera capa aplicando ReLU, y luego la capa final.

Para el proceso de entrenamiento se define la función 'train' cual recibe como argumentos la ruta al archivo con los vectores de características, la ruta hacia el archivo con las labels, y una ruta donde guardar el modelo. Así dentro de la función se cargan los vectores y se leen las labels, usando LabelEncoder, se realiza fit_transform sobre estas, ajustando el objeto de modo que genera una codificación a entero para cada clase (fit), y luego transforma las labels proporcionadas (transform), es importante guardar posteriormente el LabelEncoder, de manera futuros usos como la validación se base en las mismas codificaciones de labels, manteniendolas consistentes. El array de feature vectors y las labels codificadas se pasan a tensores, y se entregan a TensorDataset, generando un objeto que entrega sus ítems de manera 'vector, label' formato cual espera DataLoader, se inicializa estableciendo un tamaño de batch de 16, y shuffle como 'True', desordenando el dataset de manera que el entrenamiento no acostumbre a un orden de entradas específico. Se define CrossEntropyLoss como función de pérdida, Adam como optimizador, y cuda como device si está disponible. Finalmente estableciendo 10 épocas se procede a entrenar, pasando los inputs por el modelo, calculando la pérdida por época y realizando optimizer.step() para ajustar los parámetros. El modelo entrenado se guarda a la ruta indicada como argumento de la función train.

Para la validación del modelo se define la función 'val', cual recibe los mismos argumentos que la función de entrenamiento, además de una ruta donde escribir los resultados de accuracy, y del mismo modo carga los vectores y las labels, esta vez cargando además el LabelEncoder usado anteriormente, para ahora solo realizar la transformación de las labels con el fit hecho en

entrenamiento. Se inicializan TensorDataset, DataLoader y SimpleModel, se define CE Loss como función de pérdida y se pone el modelo en modo de evaluación. Pasando todas las entradas al modelo y obteniendo las predicciones, se calcula el Loss y la pérdida tanto total como por clase, esto logrado al contar las veces que el modelo predice una clase y las veces en que es correcta, esta información se va guardando en un diccionario con el nombre de las labels como clase, aunque las labels están codificadas, se pueden acceder a sus nombres a través de LabelEncoder.classes_. El accuracy por clase se transforma a numpy array, y se guarda en un archivo cuyo nombre indica el método de clasificacion y encoder usado, por ejemplo ‘ACC_MLP_CLIP.npy’

Finalmente para hacer uso de estas funciones, se definen las rutas a los archivos con los vectores de características y labels tanto como de entrenamiento como validación, estas se pasan a la función correspondiente, de modo que cuando se desee entrenar el modelo, espacio de características sea el de entrenamiento, y equivalentemente para validación.

2.4 SVM

La clasificación a través de SVM resulta más breve en cuanto a código. Al igual que en el código de la MLP, se establecen las rutas a vectores de características de validación y entrenamiento. Las funciones están implementadas de la siguiente manera.
Para el entrenamiento, se cargan los vectores y labels, usando Label Encoder se realiza ‘fit_transform’ sobre estas últimas, convirtiéndolas en enteros. Desde sklearn.svm se importa SVC una clase de python que se inicializa con kernel ‘rbf’, Radial Basis Function y probability en ‘True’. Teniendo el objeto se realiza ‘fit’ para entrenarlo, pasándole los vectores de características y las labels codificadas. Tanto el modelo entrenado como el LabelEncoder se guardan para posterior evaluación.
Para la validación de misma manera se cargan los vectores y labels correspondientes, junto al modelo entrenado y el LabelEncoder, con este se transforman las labels manteniendo consistencia en el mapeo de clases definido en el entrenamiento. Se pasan los vectores por el modelo realizando ‘model.predict(features)’, con las predicciones ya se pueden obtener las métricas de interés, importando classification_report y accuracy_score desde sklearn.metrics, se puede obtener el accuracy total y un reporte de desempeño según cada clase, entregando precisión, recall y f1-score. El accuracy por clase es calculado manualmente del mismo modo que en la MLP, este proceso queda dentro de una función llamada luego de obtenidas las predicciones, recorriendo cada una y viendo cuando la predicción concuerda con las labels de

validación, de igual modo los resultados son guardados en un archivo .npy. Los parámetros de la SVM se mantuvieron por default, considerando un C de 1, indicando la penalización de error, y un parámetro gamma en ‘scale’, siendo igual a 1 partido en la cantidad de entradas por la varianza total.

2.5 RF

La clasificación con Random Forest sigue un procedimiento similar a los anteriores, con el mismo código para definir las rutas a vectores de entrenamiento y validación. En la función de entrenamiento se cargan los vectores y labels correspondientes, los cuales se transforman a través de LabelEncoder. Al igual que SVM, el modelo se obtiene de sklearn, desde el módulo ensemble, se importa además desde sklearn.model_selection GridSearchSV, cual permite evaluar a partir de varios de un modelo, cuál combinación da el mejor resultado. Así se crea el diccionario ‘param_grid’, el cual tiene por llaves los parámetros comunes del RF, ‘n_estimators‘, ‘max_depht’ y ‘min_samples_split’. Aun así por efecto de rendimiento se establece solo un valor por cada parámetro, 200 árboles, 50 de profundidad máxima y 2 samples como mínimo para seguir realizando splits. Se crea un objeto de GridSearchSV, inicializando con RandomForestClassifier y el diccionario definido, se ajusta con ‘fit’ y obteniendo el mejor modelo con ‘grid_search.best_estimator_’ Tanto el modelo como el LabelEncoder se guardan en un archivo a una ruta entregada.
Se implementa una función para validar, cargando los vectores y labels entregadas, cuales deben ser las de validación, además cargando el modelo y el LabelEncoder. Se transforman las labels y se pasan a través del modelo junto a los vectores realizando ‘model.predict(features)’ para así obtener las predicciones. Al igual que en el código de RF, se usa classification_report y accuracy_score, además calculando el accuracy por clase del mismo modo que en casos anteriores, se guarda a un archivo .npy los resultados.

2.6 Gráfico

Se busca realizar un gráfico de barras indicando el accuracy por clase para los encoders y métodos de clasificación usados. Para ello se realiza el código ‘graph.py’, donde especificado un método de clasificación se establecen las rutas a los archivos .npy donde se escribieron los resultados de accuracy para cada encoder con el método especificado, siendo estos MLP, SVM o RF.

Cargando los numpy arrays, se construye un plot con matplotlib.pyplot, con el eje ‘x’ indicando las diferentes labels, cada una con tres barras para los tres encoders, así el eje ‘y’ indica el nivel de accuracy obtenido. Con este código se construyen tres gráficos para cada uno de los tres métodos de clasificación y tres encoders.

2.7 Confusion Matrix

En aquel método de clasificación donde se consiguen los mejores resultados, se implementa un código adicional. Importando la librería seaborn, y la función confusion_matrix desde sklearn.metrics, se construye un heatmap del resultado de pasar las labels ground truth y las predicciones a la función, con las labels en ambos ejes siendo el nombre de las clases. El resultado se muestra en pantalla y se guarda en la carpeta ‘plots’.

3. Resultados Experimentales

3.1 MLP

Epochs: 10
Batch size: 16

Tabla 3.1.1 Accuracy total por encoder

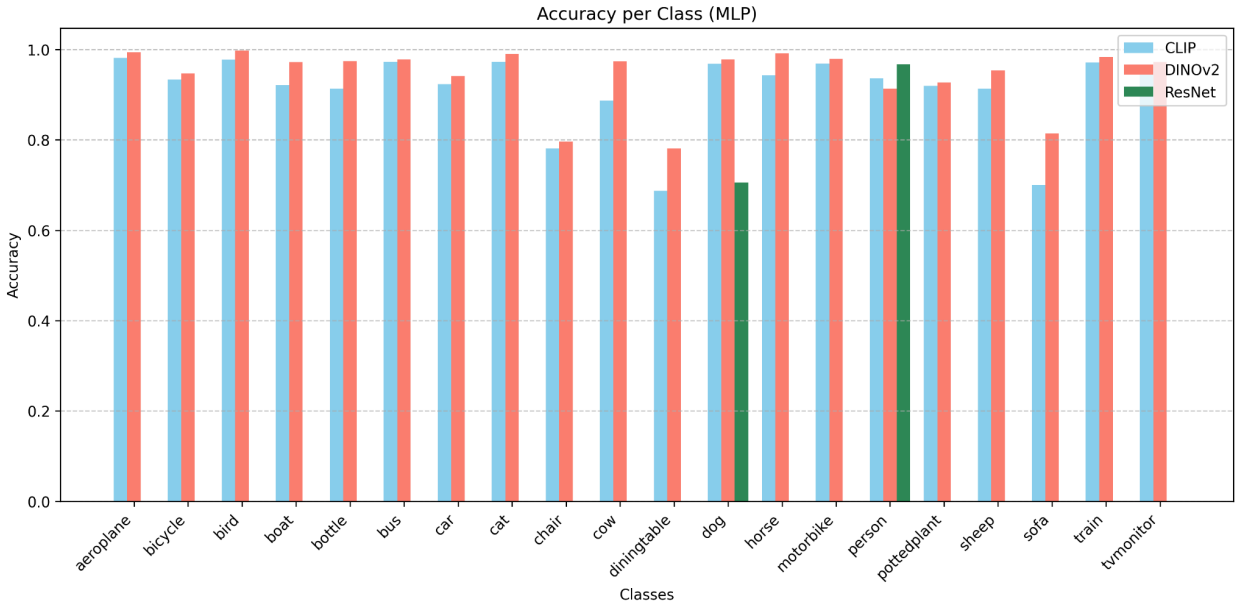
	ResNet34	DINOv2	CLIP
Accuracy total	0.2342	0.9478	0.9289

Tabla 3.1.2 Accuracy por clase

	ResNet34	DINOv2	CLIP
aeroplane	0.000	0.9939	0.9816
bicycle	0.000	0.9476	0.9333
bird	0.000	0.9972	0.9774
boat	0.000	0.9721	0.9209
bottle	0.000	0.9745	0.9133
bus	0.000	0.9785	0.9731
car	0.000	0.9412	0.9233
cat	0.000	0.9902	0.9727
chair	0.000	0.7962	0.7811
cow	0.000	0.9733	0.8867
diningtable	0.000	0.7812	0.6875
dog	0.7056	0.9780	0.9679
horse	0.000	0.9912	0.9427
motorbike	0.000	0.9792	0.9688
person	0.9673	0.9132	0.9367
pottedplant	0.000	0.9274	0.9194

sheep	0.000	0.9533	0.9133
sofa	0.000	0.8144	0.7006
train	0.000	0.9837	0.9715
tvmonitor	0.000	0.9720	0.9626

Figura 3.1.1 Accuracy por clase (MLP)



3.2 SVM

Kernel: RBF
C: 1
Gamma: ‘scale’ (1 / n_features * X.var())

Tabla 3.2.1 Accuracy total por encoder

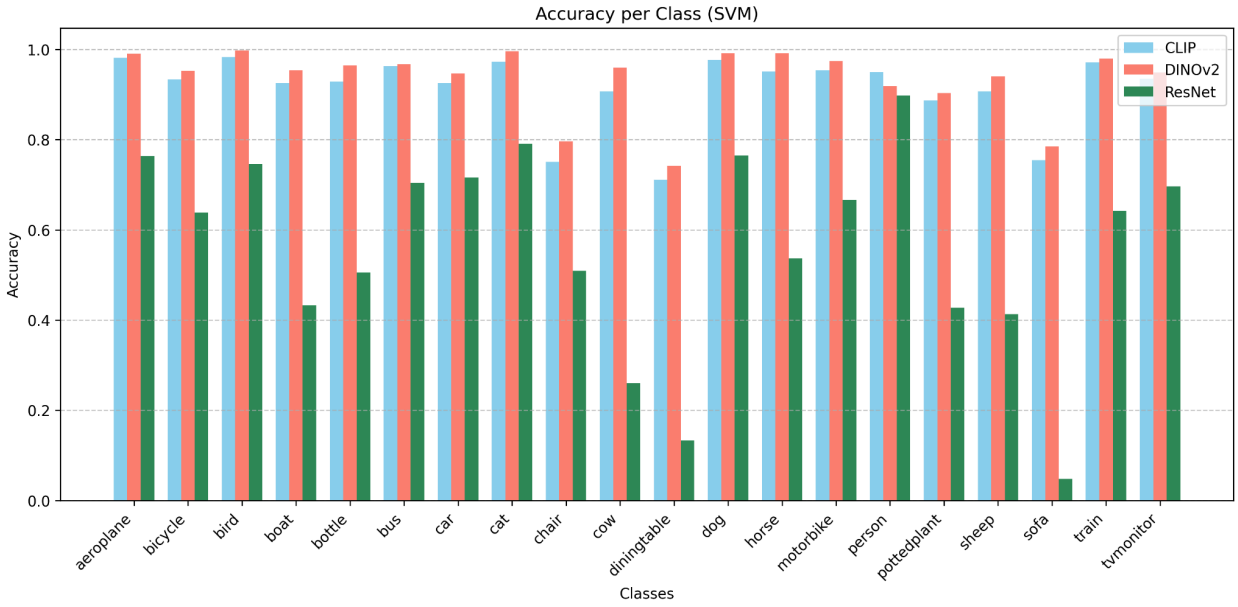
	ResNet34	DINOv2	CLIP
Accuracy total	0.6624	0.9454	0.9320

Tabla 3.2.2 Accuracy por clase

	ResNet34	DINOv2	CLIP
aeroplane	0.7638	0.9908	0.9816
bicycle	0.6381	0.9524	0.9333

bird	0.7458	0.9972	0.9831
boat	0.4326	0.9535	0.9256
bottle	0.5051	0.9643	0.9286
bus	0.7043	0.9677	0.9624
car	0.7161	0.9463	0.9258
cat	0.7910	0.9961	0.9727
chair	0.5094	0.7962	0.7509
cow	0.2600	0.9600	0.9067
diningtable	0.1328	0.7422	0.7109
dog	0.7648	0.9915	0.9763
horse	0.5374	0.9912	0.9515
motorbike	0.6667	0.9740	0.9531
person	0.8979	0.9183	0.9499
pottedplant	0.4274	0.9032	0.8871
sheep	0.4133	0.9400	0.9067
sofa	0.0479	0.7844	0.7545
train	0.6423	0.9797	0.9715
tvmonitor	0.6963	0.9486	0.9346

Figura 3.2.1 Accuracy por clase (SVM)



3.3 Random Forest

n_estimators: 200
max_depth: 50
min_samples_split: 2

Tabla 3.3.1 Accuracy total por encoder

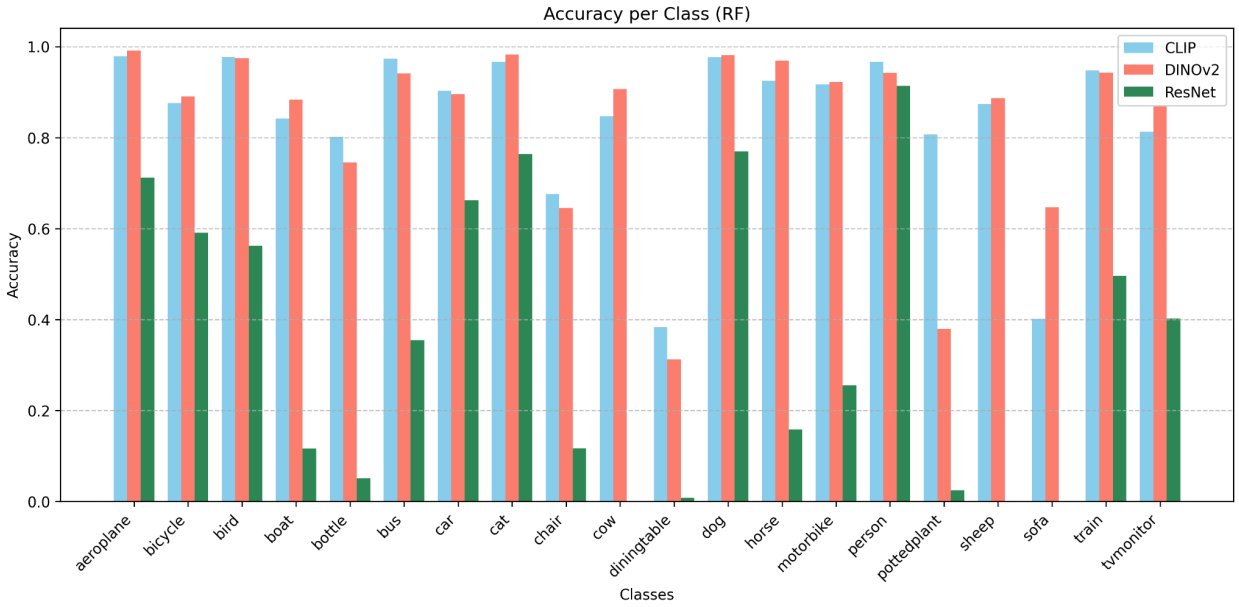
	ResNet34	DINOv2	CLIP
Accuracy total	0.5123	0.8908	0.8904

Tabla 3.3.2 Accuracy por clase

	ResNet34	DINOv2	CLIP
aeroplane	0.7117	0.9877	0.9785
bicycle	0.5905	0.9095	0.8762
bird	0.5621	0.9689	0.9774
boat	0.1163	0.8837	0.8419
bottle	0.0510	0.7449	0.8010
bus	0.3548	0.9462	0.9731
car	0.6624	0.8951	0.9028

cat	0.7637	0.9805	0.9668
chair	0.1170	0.6604	0.6755
cow	0.0000	0.8933	0.8467
diningtable	0.0078	0.3281	0.3828
dog	0.7699	0.9780	0.9763
horse	0.1586	0.9780	0.9251
motorbike	0.2552	0.9427	0.9167
person	0.9132	0.9418	0.9663
pottedplant	0.0242	0.4355	0.8065
sheep	0.0000	0.9133	0.8733
sofa	0.0000	0.6407	0.4012
train	0.4959	0.9512	0.9472
tvmonitor	0.4019	0.8458	0.8131

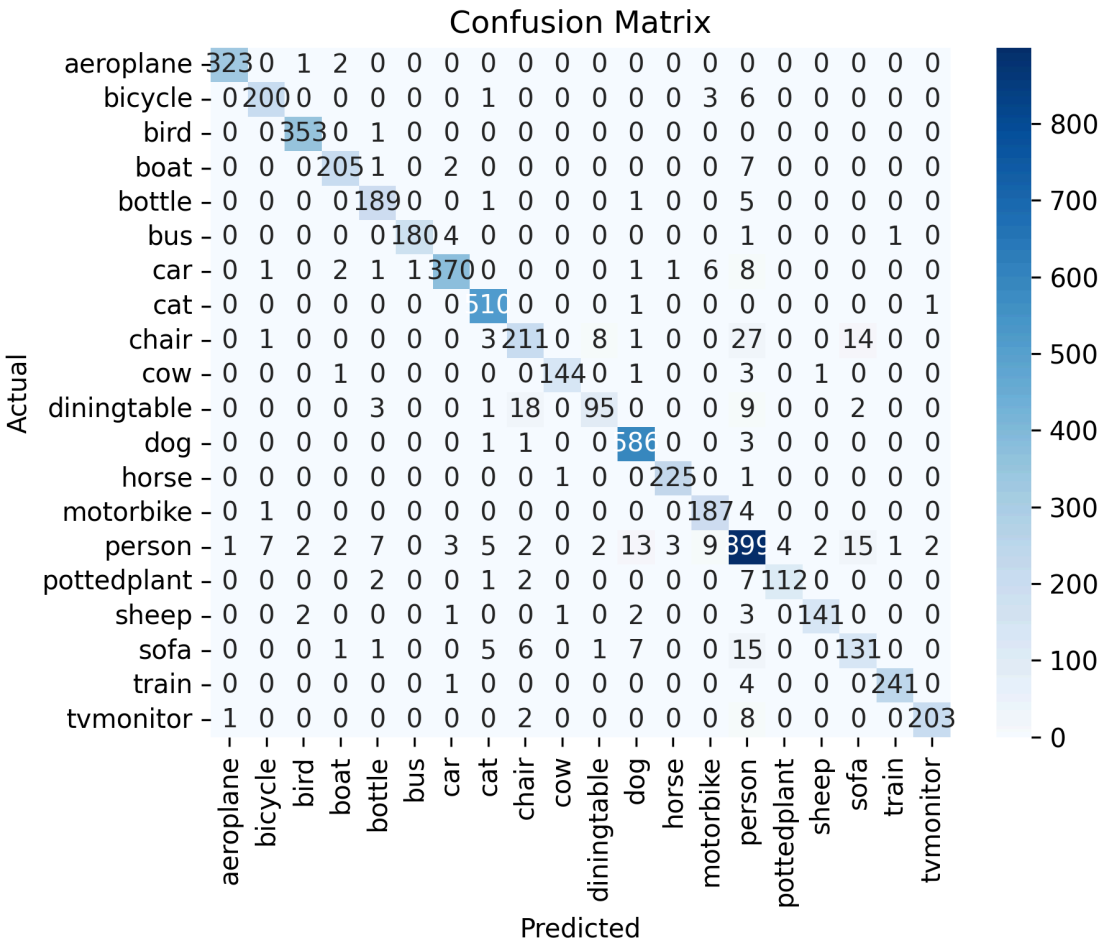
Figura 3.3.1 Accuracy por clase (RF)



3.4 Matriz de Confusión

Mejor modelo: SVM y DINOv2

Figura 3.4.1 Confusion Matrix (DINOv2 y SVM)



4. Discusión

Sobre los resultados de clasificación a través de la MLP, se puede observar que el encoder con mejor rendimiento en cuanto a accuracy fue DINOv2 con un 0.9478. Mientras que el peor fue ResNet34 con un 0.2342, muy por debajo de los otros dos encoder. Los resultados de accuracy por clase revelan como ResNet34 falla en la gran mayoría de predicciones, obteniendo un accuracy de cero en todas excepto ‘dog’ y ‘person’, siendo alto este último con un 0.9673, la razón de esto siendo que al parecer todas las predicciones fueron hacia una de estas dos clases, por lo que tendrán varios aciertos y un buen accuracy, pero en teoría una mala precisión. Los resultados de ResNet34 se pueden atribuir a la calidad de su espacio de características, los vectores extraídos no estarían a la par con encoders como DINOv2 y CLIP, demostrando un inferior rendimiento en experimentos y tareas anteriores. En cuanto a los demás encoders, se observa de las tablas como presentan en general buen desempeño, con las clases de menos accuracy en ambos siendo ‘chair’, ‘sofa’, y ‘diningtable’ con este ultimo el mas bajo con un 0.6875 en CLIP y 0.7812 en DINOv2. El gráfico de barras representa los resultados adecuadamente, pudiendo observarse como DINOv2 sobresale frente a CLIP por un margen no tan alto.

Para la clasificación con SVM, el mejor accuracy es obtenido por DINOv2 nuevamente con un 0.9454, CLIP presenta resultados similares, y ResNet34 en esta oportunidad muestra un aumento en accuracy en respecto a la MLP, con un 0.6624. Observando la tabla se aprecia como los resultados por clase son en general mejores, pero aun no cercanos a aquellos de DINOv2 o CLIP, lo máximo que alcanza es en ‘person’ con un 0.8979, mientras su mínimo es en ‘diningtable’ con 0.1328, ademas esta vez no se presentan clases con accuracy 0. En cuanto a los demás encoders, DINOv2 tiene un rendimiento menor solo por 0.0024 décimas, mientras que CLIP mejora por 0.0031. El gráfico de barras es similar, ahora con más presencia de los accuracy de ResNet34, DINOv2 sigue mostrándose mejor, con CLIP bastante cerca, se aprecia además cómo ‘sofa’, ‘chair’ y ‘diningtable’ siguen siendo las clases donde los tres modelos suelen errar.

En la clasificación por RF, se puede ver como el accuracy total en los tres encoders disminuyen en no más de 0.15, ResNet34 es el cual se ve más afectado, mientras que CLIP y DINOv2 bajan alrededor de 0.05. En este caso ResNet nuevamente presenta clases con accuracy de 0, y unas muy bajas, sobre todo en aquellos donde ya se ha visto que los modelos rinden de peor manera. En cuanto a los demás encoders, se dan unas clases en donde CLIP tiene resultados

por sobre DINOv2, como en ‘person’, ‘bus’, y ‘train’, aun así el accuracy entre ambos es en la mayoría de los casos similar. El gráfico de barras demuestra estos resultados y da a ver que la clase en donde más diferencia se produce entre los encoders es en ‘diningtable’.

El modelo con mejores resultados fue determinado por el promedio entre el accuracy de cada encoder, siendo para la MLP 0.7046, para SVM 0.8466 y para RF 0.7645. Resulta así SVM como el método de clasificación cual tuvo mejor rendimiento, así se obtiene la matriz de confusión de este con las predicciones hechas con el espacio de características de DINOv2, el cual tuvo mayor accuracy total entre los tres encoders. A partir de la matriz, se observa la relación entre las labels reales, y las predicciones, concordando con los resultados ya analizados, la clase ‘person’ es la cual más aciertos tiene, sin embargo también la cual en la que se dan más errores, con varias de las demás clases habiendo sido predichas en su lugar, por ejemplo ‘sofa’ en 15 ocasiones. Se observa cómo además la clase ‘diningtable’ es aquella con menos aciertos TP de las demás, con tan solo 95, comúnmente siendo malinterpretada por ‘chair’, sin embargo se observa que la label a su vez es aquella con menos samples, siendo 128, cuando en comparación, ‘person’ cuenta con 899 solo en sus predicciones TP.

Adicionalmente, considerando que DINOv2 es el encoder que obtuvo los mejores resultados, el accuracy que tuvo cada modelo clasificador con este encoder fue de 0.9454 para la SVM, 0.9478 para la MLP y 0.8908 para Random Forest, lo cual nos podría indicar que en algunos casos la MLP se ajusta mejor al espacio de características del encoder que una SVM, pero en estos casos habría que tomar en cuenta que implementar una SVM es más liviano a nivel de procesamiento.

5. Conclusión

Es posible observar que en base a los resultados obtenidos, entre todos los encoders el mejor fue DINOv2 seguido por CLIP y muy por debajo ResNet, incluso en algunos casos CLIP tuvo un desempeño mejor que DINOv2 con algunas clases, por lo que ambos son buenos encoders para reconocimiento de imágenes, lo cual resulta ideal para nuestro objetivo, pues al tener dos encoders con un desempeño similar, es más fácil analizar cuáles de los clasificadores funciona mejor y también nos permitió analizar si es que uno de los modelos de clasificación funciona mejor con un encoder que otro gracias al espacio de características generados por estos encoders.

Con respecto a los modelos de clasificación, el con mejor accuracy fue la MLP con el espacio de características generado por DINOv2, con 94.78%, pero en promedio, la SVM obtuvo un mejor desempeño entre todos los encoders, lo cual nos indica que el espacio de características generado por estos encoders puede ser dividido espacialmente por una SVM con bastante precisión, tomando en cuenta, que al tener 20 clases en nuestro dataset, fue necesario generar 20 hiperplanos en el espacio para subdividir estas clases. En la matriz de confusión se puede ver evidenciado el buen desempeño de la SVM junto a DINOv2, pero este gráfico también nos indica que probablemente la clase más cercana espacialmente a todas las otras es la clase “person”, pues esta se confundió algunas veces con la clase “dog” y “sofa” entre otras, lo cual nos indica que no fue posible dividirla perfectamente por hiperplanos.

Tomando en cuenta lo mencionado anteriormente, la SVM parecería ser una alternativa más segura que una MLP, considerando que la SVM pareció reflejar de manera más fiel las capacidades de extraer características de cada encoder, mientras que la MLP demostró tener un desempeño marginalmente mayor (inferior al 1%). Por otro lado, Random Forest demostró ser aproximadamente un 10% peor que la SVM, por lo que su uso sólo se vería justificado en casos en los que se busca interpretabilidad en la decisión del modelo.

En conclusión, el encoder con mejor desempeño fue DINOv2 seguido por CLIP y el modelo de clasificación que mejor trabajó con todos los encoders fue la SVM, aunque la MLP obtuvo los mejores resultados con DINOv2, por lo que el modelo clasificador a utilizar va a depender totalmente del espacio de características y de la complejidad de lo que se desee clasificar.