Project 1

Modelo de Clasificación de Imágenes con MLP

Daniel Alejandro Torres González 1202533

Juan David Plata Garrido 1202509

Deep Learning TEL B

Profesor Diego Renza Torres

Ingeniería en Multimedia

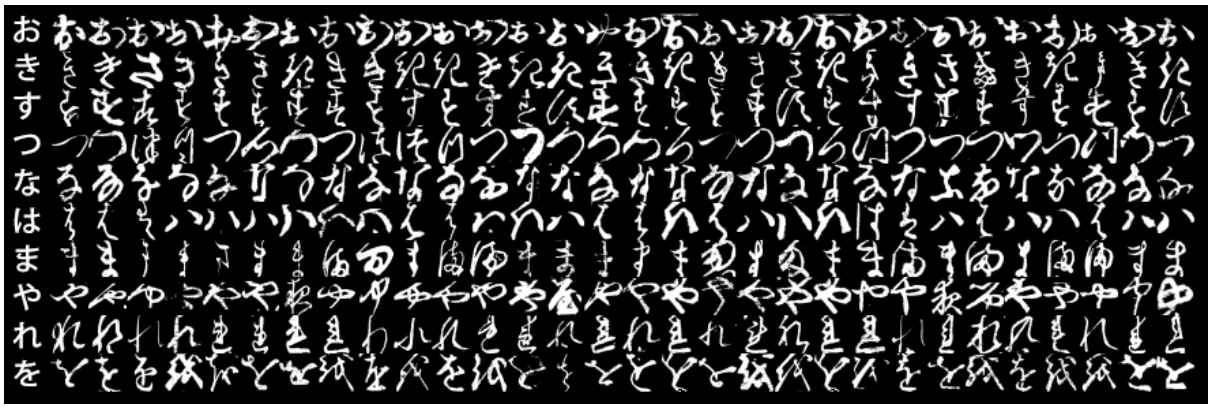
Universidad Militar Nueva Granada

2025-1

Modelo de Clasificación de Imágenes con MLP

Para el desarrollo del proyecto se utilizó un dataset de tipo MNIST llamado Kuzushiji 49, el cual tiene 49 clases de 28x28 en escala de grises con 270.912 imágenes.

k49-[train/test]-[images/labels].npz: Los archivos estaban en formato npz, estos son arreglos de numpy, por defecto venían 232.365 datos en train y 38.547 en test. Posteriormente se asignaron otros 38.547 para los datos de validación.



Finalmente, los arreglos quedaron de la siguiente forma:

Tamaño de datos de entrenamiento después de la división: (193818, 28, 28, 1)

Tamaño de datos de validación: (38547, 28, 28, 1)

Posteriormente se pasaron los labels se pasaron a tipo One-Hot el cual sirve para para convertir datos categóricos en un formato numérico que las redes neuronales puedan procesar.

Dimensión de train_labels (One-Hot): (193818, 49)

Dimensión de val_labels (One-Hot): (38547, 49)

Dimensión de test_labels (One-Hot): (38547, 49)

Este se utiliza debido a que las redes neuronales funcionan mejor con datos numéricos, y el One-Hot permite representar categorías de manera que la red pueda aprender patrones sin sesgo de orden. Un ejemplo de esto seria

Rojo, **Verde**, **Azul**

Rojo → [1, 0, 0]

Verde → [0, 1, 0]

Azul → [0, 0, 1]

A diferencia de asignar números enteros (ej. Rojo=1, Verde=2, Azul=3), no sugiere una relación de magnitud u orden.

Se eligieron los siguientes hiperparámetros para la selección automática de estos:

- Número de capas ocultas: Se seleccionó un rango entre 1 y 5 capas ocultas.
- Neuronas por capa: Se seleccionó un rango entre 32 y 512 neuronas con un salto de 32 por neuronas.
- Método de activación: Selección entre ReLU, tanh, sigmoid y softmax
- Método de inicialización: Selección entre he_uniform, glorot_uniform, glorot_normal.
- Learning rate: Rango entre 1e-4 y 1e-2

Se uso Keras Tuner para la selección automática de hiperparámetros, para esto se realizaron 10 pruebas con 10 épocas para determinar cuáles son los hiperparámetros más eficientes para obtener la mejor precisión de validación.

```
Trial 10 Complete [00h 03m 34s]
val_accuracy: 0.37331050634384155

Best val_accuracy So Far: 0.929566502571106
Total elapsed time: 00h 40m 41s

Mejores hiperparámetros encontrados:
- Número de capas ocultas: 2
- Neuronas por capa: [512, 448]
- Activación: relu
- Inicializador: glorot_normal
- Learning rate: 0.00016960331830446725
```

Estos son los mejores hiperparámetros encontrados luego de realizar las 10 pruebas, una vez obtenidos los hiperparámetros se entrena el modelo.

El modelo cuenta con la capa de entrada, luego una capa flatten que convierte la matriz de la imagen en un vector para que pueda ser trabajada posteriormente en las 2 capas fully connected, con 512 y 488 neuronas respectivamente, la capa de salida tiene 49 neuronas debido a que es la cantidad de clases del dataset.

```
history = best_model.fit(
    train_images, train_labels,
    epochs=20,
    batch_size=64,
    validation_data=(val_images, val_labels),
    callbacks=[early_stop]
)
```

Para el entrenamiento del modelo se utilizó un número de 20 épocas y un batch size de 64, en validation data se usaron los arreglos de validación creados previamente, también para los callbacks se utilizó early stop el cual detiene el entrenamiento si se repite varias veces el mismo valor en el error. Esta es otra forma de reducir el overfitting

```
Mejores hiperparámetros encontrados:
- Número de capas ocultas: 2
- Neuronas por capa: [512, 448]
- Activación: relu
- Inicializador: glorot_normal
- Learning rate: 0.00016960331830446725

Epoch 1/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 14s 4ms/step - accuracy: 0.5539 - loss: 1.8190 - val_accuracy: 0.8078 - val_loss: 0.7427
Epoch 2/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 17s 3ms/step - accuracy: 0.7927 - loss: 0.7850 - val_accuracy: 0.8566 - val_loss: 0.5357
Epoch 3/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.8412 - loss: 0.5899 - val_accuracy: 0.8797 - val_loss: 0.4453
Epoch 4/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step - accuracy: 0.8652 - loss: 0.4963 - val_accuracy: 0.8931 - val_loss: 0.3943
Epoch 5/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step - accuracy: 0.8811 - loss: 0.4277 - val_accuracy: 0.9032 - val_loss: 0.3590
Epoch 6/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.8927 - loss: 0.3865 - val_accuracy: 0.9113 - val_loss: 0.3281
Epoch 7/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9003 - loss: 0.3513 - val_accuracy: 0.9154 - val_loss: 0.3122
Epoch 8/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9104 - loss: 0.3171 - val_accuracy: 0.9182 - val_loss: 0.3000
Epoch 9/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9152 - loss: 0.2990 - val_accuracy: 0.9217 - val_loss: 0.2884
Epoch 10/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9220 - loss: 0.2718 - val_accuracy: 0.9237 - val_loss: 0.2792
Epoch 11/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9242 - loss: 0.2595 - val_accuracy: 0.9262 - val_loss: 0.2719
Epoch 12/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9303 - loss: 0.2400 - val_accuracy: 0.9275 - val_loss: 0.2699
Epoch 13/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9330 - loss: 0.2277 - val_accuracy: 0.9268 - val_loss: 0.2652
Epoch 14/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9363 - loss: 0.2175 - val_accuracy: 0.9296 - val_loss: 0.2598
Epoch 15/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step - accuracy: 0.9395 - loss: 0.2062 - val_accuracy: 0.9320 - val_loss: 0.2561
Epoch 16/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9414 - loss: 0.1950 - val_accuracy: 0.9315 - val_loss: 0.2545
Epoch 17/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9431 - loss: 0.1877 - val_accuracy: 0.9317 - val_loss: 0.2535
Epoch 18/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.9468 - loss: 0.1767 - val_accuracy: 0.9329 - val_loss: 0.2492
Epoch 19/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 19s 3ms/step - accuracy: 0.9477 - loss: 0.1717 - val_accuracy: 0.9342 - val_loss: 0.2493
Epoch 20/20
3029/3029 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9512 - loss: 0.1619 - val_accuracy: 0.9346 - val_loss: 0.2508
```

Se dejó el modelo con 20 épocas debido a que no se generaba overfitting, esto daba una precisión de 95.12% y una precisión de validación de 93.42% esto da una diferencia de 1.7% por lo tanto no hay overfitting, se hicieron pruebas con 30 épocas, pero se generaba un overfitting de 2.5%.

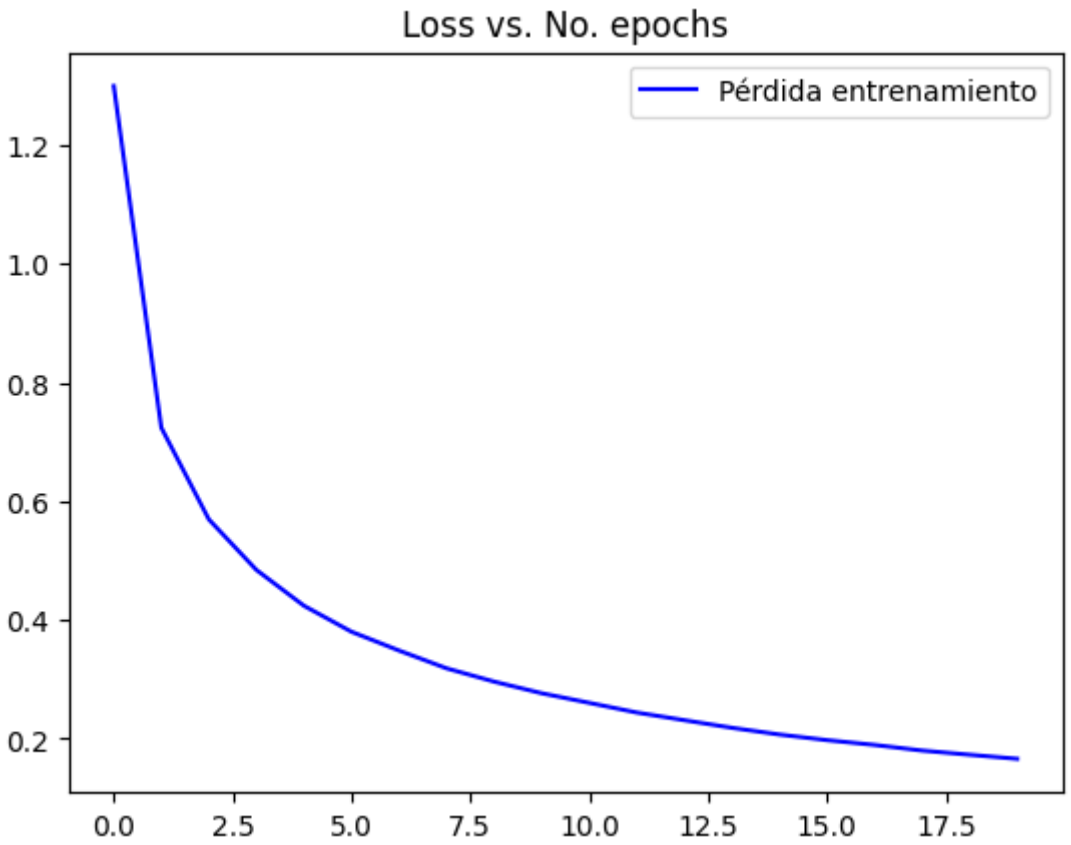
Para la mayor diminución de overfitting se usó un dropout el cual desactiva entre 20 y 30 % de las neuronas de las capas ocultas.

```
# Evaluar el modelo en el conjunto de prueba
test_loss, test_acc = best_model.evaluate(test_images, test_labels)
print(f'\nPrecisión en el conjunto de prueba: {test_acc * 100:.2f}%')
```

```
1205/1205 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8814 - loss: 0.4605

Precisión en el conjunto de prueba: 88.03%
```

Al evaluar el modelo con el conjunto de prueba, se obtuvo una precisión del 88.03%



Esta gráfica muestra como la perdida disminuye a lo largo de las épocas de entrenamiento, se usó la función de pérdida categorical_crossentropy debido a que la capa de salida usa softmax, Softmax convierte las salidas del modelo en probabilidades para cada clase, asegurando que la suma de todas las probabilidades sea igual a 1. categorical_crossentropy compara la distribución de probabilidades predicha con la verdadera (representada mediante *one-hot encoding*).