Tarea N°1:

Redes Neuronales

Cristóbal Tomás David Alberto Lagos Valtierra
Escuela de Ingeniería, Universidad de O'Higgins
28 de Octubre del 2023

***Abstract*—El objetivo de este informe es utilizar redes neuronales en un problema de clasificación de dígitos. Se utilizará el conjunto de datos Optical Recognition of Handwritten Digits Data Set. Este conjunto tiene 64 características, con 10 clases y 5620 muestras en total.**

Las redes a ser entrenadas tendrán la siguiente estructura: capa de entrada de dimensionalidad 64 (correspondiente a los datos de entrada), capas ocultas (una o dos) y capa de salida con 10 neuronas y función de activación softmax. La función de loss (pérdida) es entropía cruzada. El optimizador que se usará es Adam.

Se usará la biblioteca PyTorch para entrenar y validar la red neuronal que implementa el clasificador de dígitos. Se analizarán los efectos de cambiar el tamaño de la red (número de capas ocultas y de neuronas en estas capas) y la función de activación.

Con el objetivo de evitar sobreajuste en el entrenamiento de la red neuronal, se implementarán estrategias con el fin de evitar este comportamiento, de esta forma, deteniendo oportunamente el entrenamiento sin comprometer la correcta implementación del modelo.

Una vez entrenados los modelos se tomará aquella red con mayor precisión en el conjunto de validación y se procederá a calcular la matriz de confusión normalizada con su respectiva precisión normalizada.

I. Introduction

En la actualidad, el aprendizaje automático y las redes neuronales se han consolidado como herramientas poderosas para resolver una diversidad de problemas, que van desde la simple clasificación de números escritos a mano hasta desafíos más complejos como el diagnóstico médico y la conducción autónoma. Este informe se adentra en el campo del aprendizaje automático al utilizar redes neuronales para abordar un problema de clasificación de dígitos, centrándose en el conjunto de datos Optical Recognition of Handwritten Digits. Aunque pueda parecer una tarea sencilla, esta tiene un impacto significativo en la tecnología actual y en diversas áreas de la ciencia y la industria.

La precisión en el entrenamiento de modelos de aprendizaje automático es un elemento crítico en la investigación y la implementación exitosa de sistemas inteligentes. Cuando se aplican en situaciones del mundo real, las redes neuronales pueden ofrecer resultados excepcionales, siempre y cuando se ajusten y optimicen adecuadamente.

El entrenamiento preciso de modelos de clasificación de dígitos se convierte en un componente fundamental en ciertas aplicaciones, como lo puede ser el salvaguardar la integridad de las transacciones financieras. En este informe, exploraremos cómo diferentes configuraciones de redes neuronales, que incluyen el número de capas ocultas, la cantidad de neuronas y la función de activación, influyen en la precisión del modelo. Además, se aplicarán estrategias para evitar el sobreajuste, lo que garantizará que el modelo se adapte de manera óptima a los datos.

Al comprender la relevancia de entrenar modelos de manera precisa y aplicar estrategias para asegurar que las redes neuronales se ajusten de la mejor manera a los datos, se entenderán como desarrollar sistemas de reconocimiento de caracteres escritos a mano más efectivos y confiables. La evaluación y selección de la mejor configuración de la red pueden tener un impacto directo en la calidad de las aplicaciones que dependen de estos sistemas, mejorando así la precisión y la eficiencia en una amplia gama de aplicaciones como lo son el procesamiento postal, la industria bancaria entre muchos otros.

II. Marco Teórico

A. Machine Learning

Machine Learning es una rama de la inteligencia artificial que se enfoca principalmente en desarrollar algoritmos y modelos que permiten a las computadoras aprender y mejorar su desempeño en tareas específicas a través de la experiencia, sin la necesidad de ser programadas de manera explícita.

En particular, el Aprendizaje Supervisado permite que la máquina aprenda de manera explícita a partir de los datos proporcionados. Además de los datos de entrada, estos van acompañados de una etiqueta con la clase a la que corresponde, es decir, los datos de salida. A medida que va aprendiendo se le informa al modelo si realizó una correcta asignación a la clase esperada con el objetivo de ajustar los parámetros y mejorar en las métricas que evalúan que tan bien realizó una asignación.

B. Aplicaciones

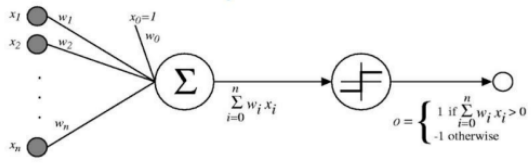
El aprendizaje supervisado tiene una amplia gama de aplicaciones en diversos campos. En este enfoque de aprendizaje, los modelos se entrenan utilizando datos etiquetados, lo que significa que se les proporcionan ejemplos de entrada junto con las salidas deseadas correspondientes. A continuación, se presentan algunas de las aplicaciones más comunes del aprendizaje supervisado con redes neuronales:

1) Diagnóstico Médico: Las redes neuronales se utilizan en la interpretación de imágenes médicas, como radiografías, resonancias magnéticas y tomografías computarizadas. Ayudan en la detección temprana de enfermedades y en la identificación de patrones anómalos.

2) Detección de Fraudes: En el ámbito financiero, las redes neuronales se emplean para detectar actividades fraudulentas. Esto incluye la identificación de transacciones fraudulentas con tarjetas de crédito y la prevención de fraudes en línea.
3) Conducción Autónoma: En la industria automotriz, las redes neuronales se utilizan en vehículos autónomos para reconocer objetos y tomar decisiones de conducción seguras en tiempo real.

C. Redes Neuronales

Las redes neuronales son un tipo de modelo aplicable al campo de aprendizaje supervisado inspirado en el funcionamiento del cerebro humano, en particular, a las redes de neuronas presentes en el cerebro. Este modelo está diseñado para procesar información de manera similar a como lo hacen las neuronas en el cerebro, esto lo hace implementando el modelo conocido como Perceptrón.



$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Fig. 1: Neurona y perceptrón.

El perceptrón toma un conjunto de entradas (características) ponderadas y las suma. Luego, aplica una función de activación (generalmente una función escalón o similar) al resultado de la suma ponderada. Si el valor resultante cumple cierto umbral, la neurona se activa y produce una salida; de lo contrario, no produce ninguna salida.

Las funciones de activación juegan un papel fundamental en la introducción de no linealidad en el modelo. Dos de las funciones de activación más comunes son la función ReLU (Rectified Linear Unit) y la función tanh (tangente hiperbólica). A continuación, se describe cada una:

- ReLU: La función ReLU es una función de activación no lineal que es cero para todos los valores de entrada negativos y lineal para valores positivos. En otras palabras, si la entrada es negativa, la salida es cero; de lo contrario, la salida es igual a la entrada. Es conocida por su simplicidad y eficiencia computacional, ya que es fácil de calcular y no sufre de problemas de desvanecimiento del gradiente.
- Tanh: La función tanh es similar a la función sigmoide en términos de forma, pero su rango de salida es [-1, 1]. Esto significa que la función tanh puede producir valores negativos, lo que la hace más adecuada para problemas en los que las entradas tienen una media cercana a cero. Se utiliza comúnmente en redes neuronales para tareas que requieren una salida centrada en cero, como el procesamiento del lenguaje natural (NLP) y la modelización de series temporales.

La información fluye a través de la red desde la capa de entrada, donde se ingresan los datos, hasta la capa de salida, donde se produce la respuesta o predicción del modelo. Entre la capa de entrada y la de salida, pueden existir una o varias capas intermedias, llamadas capas ocultas.

Para entrenar las Redes Neuronales necesitamos hacer uso de 2 algoritmos clave en el proceso de entrenamiento del modelo conocidos como el "forward pass" y el "backward pass".

El "forward pass" y el "backward pass" son dos etapas clave en el entrenamiento de redes neuronales, particularmente en el contexto del aprendizaje supervisado y el proceso de retropropagación (backpropagation). Estas etapas son fundamentales para que una red neuronal aprenda a partir de los datos y mejore su rendimiento. Aquí se explica cada una de ellas:

- Forward Pass: es la primera etapa en el proceso de entrenamiento de una red neuronal. Durante esta fase, los datos de entrada se propagan a través de la red desde la capa de entrada hasta la capa de salida para realizar una predicción. Este proceso se repite para cada ejemplo de entrenamiento en el conjunto de datos, y la función de pérdida se acumula a medida que se procesan los ejemplos. El objetivo es minimizar esta función de pérdida para que las predicciones se acerquen cada vez más a las etiquetas reales.
- Backward Pass: es la etapa en la que los gradientes de la función de pérdida se propagan hacia atrás a través de la red neuronal, lo que permite ajustar los pesos de la red para minimizar la pérdida. Es esencial para el entrenamiento de redes neuronales, ya que permite que la red aprenda a partir de sus errores y mejore sus predicciones a medida que se ajustan los pesos. El proceso se repite a lo largo de múltiples iteraciones hasta que la red converja a una solución que minimice la función de pérdida y haga predicciones precisas en datos no vistos.

El proceso de entrenamiento en las redes neuronales es iterativo. Este proceso de reentrenamiento de la red neuronal se repite hasta que se cumpla uno de dos criterios de detención: un número fijo de épocas (pasada completa a través de todo el conjunto de datos de entrenamiento), se haya completado o la pérdida se minimice o estabilice. En el primer caso, se define un límite de épocas preestablecido y el entrenamiento continúa hasta que se alcance ese límite. En el segundo caso, se supervisa la pérdida en cada iteración del entrenamiento y se detiene cuando se alcanza un valor de pérdida aceptable o cuando la pérdida deja de disminuir de manera significativa. Estos criterios de detención son fundamentales para evitar el sobreajuste.

D. Sobreajuste

Fenómeno que ocurre cuando un modelo de machine learning se ajusta demasiado a los datos de entrenamiento, perdiendo generalidad y la capacidad de aplicar lo aprendido en nuevos conjuntos de datos. El modelo se vuelve demasiado

complejo y capaz de capturar el ruido y las particularidades de los datos de entrenamiento.

Algunos comportamientos que permiten diagnosticar cuando se está sufriendo Sobreajuste son los siguientes:

1) Pérdida de generalización: A pesar de que el modelo se desempeña muy bien en el conjunto de entrenamiento, su rendimiento en datos no vistos, como un conjunto de validación o pruebas, disminuye significativamente.
2) Alta varianza: El modelo es altamente sensible a pequeñas variaciones en los datos de entrenamiento, lo que puede llevar a resultados inconsistentes.
3) Modelo demasiado complejo: El modelo puede tener demasiados parámetros o capas, lo que lo hace altamente flexible y capaz de adaptarse a incluso ruido aleatorio en los datos de entrenamiento.

Para abordar el sobreajuste, se pueden aplicar diversas técnicas, como las siguientes:

1) Validación cruzada: La validación cruzada es una técnica que divide los datos en múltiples conjuntos de entrenamiento y validación, lo que permite evaluar el rendimiento del modelo en diferentes subconjuntos de datos.
2) Early stopping: Consiste en detener el entrenamiento del modelo antes de que alcance un número fijo de épocas (iteraciones) cuando se observa que el rendimiento del modelo en un conjunto de validación deja de mejorar o empeora.

E. Matriz de Confusión

La matriz de confusión es una tabla que se utiliza para evaluar el rendimiento de un modelo de clasificación. Muestra la relación entre las predicciones del modelo y las etiquetas reales en un conjunto de datos.

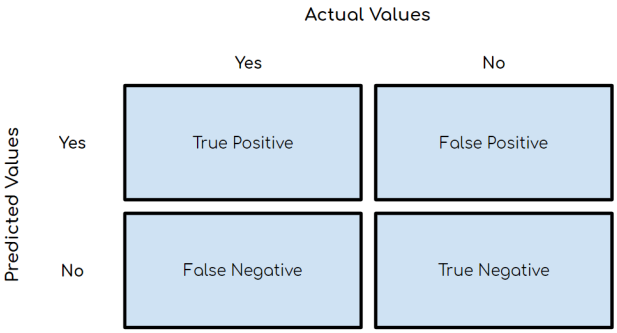


Fig. 2: Matriz de confusión.

La matriz se divide en cuatro partes principales:

- Verdaderos positivos (True Positives, TP): Representa los casos en los que el modelo predijo correctamente una clase positiva.
- Verdaderos negativos (True Negatives, TN): Representa los casos en los que el modelo predijo correctamente una clase negativa.
- Falsos positivos (False Positives, FP): Representa los casos en los que el modelo predijo incorrectamente una clase positiva cuando la verdadera clase era negativa.
- Falsos negativos (False Negatives, FN): Representa los casos en los que el modelo predijo incorrectamente una clase negativa cuando la verdadera clase era positiva.

Gracias al uso de esta matriz es que se puede evaluar el rendimiento del modelo en términos de precisión, sensibilidad, especificidad, entre otros.

F. Precisión

La precisión (accuracy en inglés) es una métrica que se calcula a partir de la matriz de confusión. Representa la proporción de predicciones correctas en relación con el número total de predicciones.

La fórmula asociada es la siguiente:

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

Fig. 3: Accuracy.

Es una métrica útil para evaluar el rendimiento general de un modelo de clasificación. Cuanto más alto sea el valor de accuracy, mejor será el rendimiento del modelo en términos de clasificar correctamente las muestras.

III. Metodología

A. Características de los Datos

Los datos a utilizar corresponden al conjunto de datos Optical Recognition of Handwritten Digits Data Set. Haciendo una división para el conjunto de entrenamiento y en el conjunto de prueba. Cada fila dentro de estas matrices posee ciertas características asociadas a los valores "featn" con "n" valores desde el 0 al 63, y una clase asociada con la etiqueta correspondiente al número en la columna "class".

Hay un total de 5620 muestras tomando en consideración ambos archivos, con un total de 65 columnas asociadas.

B. Preprocesamiento de los datos

Antes de realizar el entrenamiento de las redes neuronales debemos preparar nuestros datasets de tal forma que formemos los conjuntos de entrenamiento, validación y prueba. Para esto, dividiremos el conjunto de datos con los valores de prueba en dos subconjuntos, de esta forma, el 30% de los datos se asignan al conjunto de validación y el 70% restante al conjunto de prueba.

Una vez teniendo nuestros conjuntos, normalizamos las características presentes en los conjuntos. Esto se hace utilizando un objeto StandardScaler, que calcula las medias y desviaciones estándar de las características en df_train y luego aplica esta normalización a los otros dos conjuntos.

```
#normalización de datasets
scaler = StandardScaler().fit(df_train.iloc[:,0:64])
df_train.iloc[:,0:64] = scaler.transform(df_train.iloc[:,0:64])
df_val.iloc[:,0:64] = scaler.transform(df_val.iloc[:,0:64])
df_test.iloc[:,0:64] = scaler.transform(df_test.iloc[:,0:64])
```

Fig. 4: Normalización.

Creamos tres conjuntos de datos separados, dataset_train, dataset_val, y dataset_test, donde cada uno contiene diccionarios con dos claves: "features" y "labels". Los datos se organizan de esta manera para facilitar su uso con PyTorch y los dataloaders.

```
# Crear datasets
feats_train = df_train.to_numpy()[:,0:64].astype(np.float32)
labels_train = df_train.to_numpy()[:,64].astype(int)
#dataset de entrenamiento
dataset_train = [ {"features":feats_train[i,:], "labels":labels_train[i]} for i in range(feats_train.shape[0]) ]

feats_val = df_val.to_numpy()[:,0:64].astype(np.float32)
labels_val = df_val.to_numpy()[:,64].astype(int)
#dataset de validación
dataset_val = [ {"features":feats_val[i,:], "labels":labels_val[i]} for i in range(feats_val.shape[0]) ]

feats_test = df_test.to_numpy()[:,0:64].astype(np.float32)
labels_test = df_test.to_numpy()[:,64].astype(int)
#dataset de prueba
dataset_test = [ {"features":feats_test[i,:], "labels":labels_test[i]} for i in range(feats_test.shape[0]) ]
```

Fig. 5: Diccionarios.

Creamos tres dataloaders (dataloader_train, dataloader_val, y dataloader_test) utilizando la funcionalidad proporcionada por PyTorch. Estos dataloaders se utilizan para cargar los datos en lotes durante el entrenamiento, validación y prueba de un modelo de aprendizaje automático.

```
# Crear dataloaders
dataloader_train = torch.utils.data.DataLoader(dataset_train, batch_size=128, shuffle=True, num_workers=0)
dataloader_val = torch.utils.data.DataLoader(dataset_val, batch_size=128, shuffle=True, num_workers=0)
dataloader_test = torch.utils.data.DataLoader(dataset_test, batch_size=128, shuffle=True, num_workers=0)
```

Fig. 6: Dataloaders.

C. Entrenamiento de Modelos

Una vez realizado todo el procesamiento requerido, nos disponemos a definir una función que será la encargada de entrenar los modelos. Esta función recibirá como input el modelo a entrenar, y se encargará de entrenar el modelo evitando sobreajuste, imprimiendo gráficas que muestren métricas asociadas tanto a la matriz de confusión y como la precisión.

Para empezar, por cada época en la que se entrena el modelo haciendo uso del conjunto de entrenamiento usando batches (conjunto de muestras o ejemplos que se procesan juntos en paralelo, ya que estamos entrenando el modelo haciendo uso de GPU para mayor eficiencia), calculamos la pérdida y la precisión del modelo en cada batch y actualizamos los parámetros del modelo para mejorar su rendimiento.

```
for i, data in enumerate(dataloader_train, 0):
    # Process the current batch
    inputs = data["features"].to(device)
    labels = data["labels"].to(device)
    optimizer.zero_grad()
    outputs = model_to_train(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
    train_loss += loss.item()

    # Calcula la precisión en entrenamiento
    _, predicted = torch.max(outputs, 1)
    total_train += labels.size(0)
    correct_train += (predicted == labels).sum().item()
    all_predictions_train.extend(predicted.cpu().numpy())
    all_labels_train.extend(labels.cpu().numpy())
```

Fig. 7: Entrenamiento.

Una vez pasado el conjunto de entrenamiento, nos dispondremos a calcular la precisión y la matriz de confusión (normalizada), estos estarán asociados a los resultados de las predicciones en el conjunto de entrenamiento.

```
val_accuracy = 100 * correct_val / total_val

# Mueve las predicciones y etiquetas a NumPy arrays
predicted_gpu = np.array(all_predictions_val)
labels_gpu = np.array(all_labels_val)

# Calcula la matriz de confusión en la GPU
confusion_matrix_val = confusion_matrix(labels_gpu, predicted_gpu, normalize="true")
```

Fig. 8: Cálculo de matriz de confusión y precisión en el conjunto de entrenamiento.

Hacemos lo mismo para el conjunto de validación:

```
model_to_train.eval()
with torch.no_grad():
    val_loss = 0.0
    correct_val = 0
    total_val = 0

    for i, data in enumerate(dataloader_val, 0):
        inputs = data["features"].to(device)
        labels = data["labels"].to(device)
        outputs = model_to_train(inputs)
        loss = criterion(outputs, labels)
        val_loss += loss.item()

        # Calcula la precisión en validación
        _, predicted = torch.max(outputs, 1)
        total_val += labels.size(0)
        correct_val += (predicted == labels).sum().item()
        all_predictions_val.extend(predicted.cpu().numpy())
        all_labels_val.extend(labels.cpu().numpy())

val_accuracy = 100 * correct_val / total_val

# Mueve las predicciones y etiquetas a NumPy arrays
predicted_gpu = np.array(all_predictions_val)
labels_gpu = np.array(all_labels_val)

# Calcula la matriz de confusión en la GPU
confusion_matrix_val = confusion_matrix(labels_gpu, predicted_gpu, normalize="true")
```

Fig. 9: Cálculo de matriz de confusión y precisión en el conjunto de validación.

Con el fin de evitar el sobreajuste, se implementará la estrategia conocida como "early stopping". Es encargada de Implementar el detenimiento temprano del entrenamiento si la pérdida de validación deja de mejorar durante el reentraniemnto del modelo.

```
# Comprueba si el loss de validación aumentó y el de entrenamiento bajó
if val_loss > prev_val_loss and train_loss < prev_train_loss:
    print('Deteniendo el entrenamiento en la época %d, debido al aumento en el loss de validación y la disminución en el loss de entrenamiento.' % (epoch))
    break
```

Fig. 10: Early stopping.

IV. Resultados

Teniendo toda la lógica implementada para el entrenamiento de modelos, procederemos a crear distintos modelos con variedad de capas ocultas y funciones de activación, con el fin de determinar cuál de todos los modelos propuestos a entrenar se desempeña mejor según las métricas de accuracy.

De todos los modelos entrenados, aquel que tuvo mejores resultados fue aquel que estaba formado por 2 capas ocultas con 40 neuronas cada una y función de activación ReLU.

Al graficar la pérdida de validación y entrenamiento en el paso del tiempo obtenemos lo siguiente:

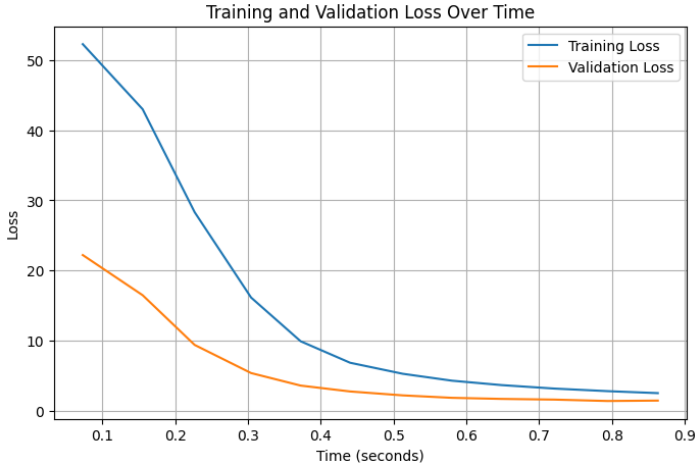


Fig. 11: Pérdida de validación y entrenamiento en el paso del tiempo .

Al calcular la matriz de confusión en el conjunto de entrenamiento obtenemos lo siguiente:

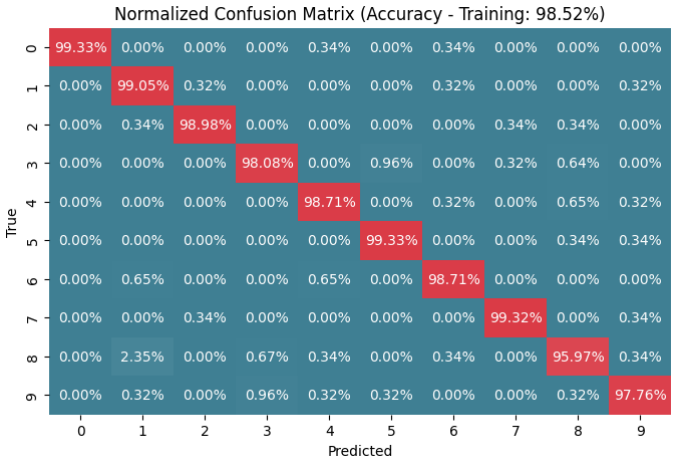


Fig. 12: Matriz de confusión en el conjunto de entrenamiento.

Al calcular la matriz de confusión en el conjunto de validación obtenemos lo siguiente:

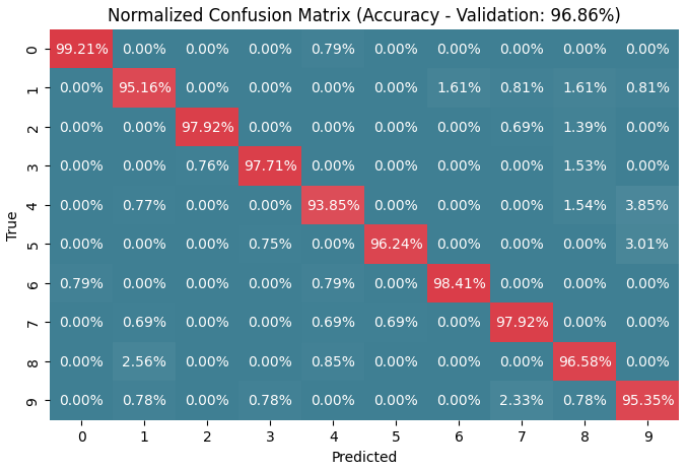


Fig. 13: Matriz de confusión en el conjunto de validación.

Al calcular la matriz de confusión en el conjunto de prueba obtenemos lo siguiente:

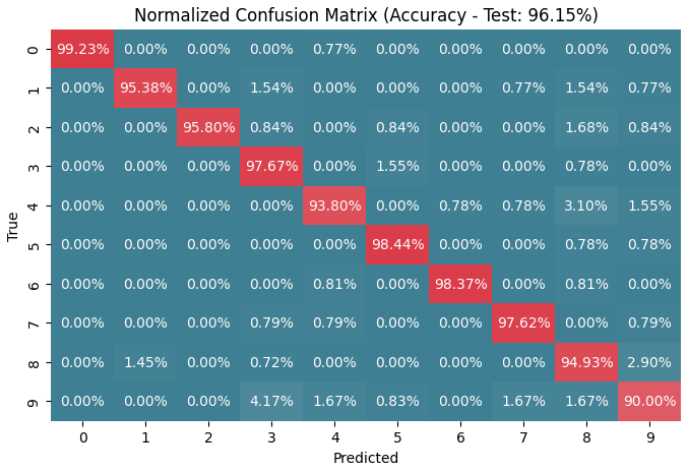


Fig. 14: Matriz de confusión en el conjunto de prueba.

V. Análisis

Al ver la pérdida en el conjunto de entrenamiento versus el de validación, podemos observar que en una primera instancia, es mayor en el conjunto de entrenamiento, sin embargo, a medida que va avanzando se va igualando al de validación hasta el punto en que se cumple que se observa que el rendimiento del modelo en un conjunto de validación deja de mejorar.

En cuanto a los resultados obtenidos a partir de las matrices de confusión podemos observar que el número con mejores resultados asociados a correctas predicciones fue el número 0. Tal como se puede apreciar en las distintas matrices, obtuve un rendimiento mayor al 99% en los conjuntos de entrenamiento, prueba y validación.

Por otra parte, el dígito con el menor porcentaje de precisión obtenido fue el número 9, lo cual puede tener bastante sentido por su similitud con el dígito 3 que fue el número que se llevó casi 4% de predicciones, al verlos en la tabla del conjunto de prueba.

VI. Conclusiones Generales

En este informe, se aplicaron redes neuronales al problema de clasificación de dígitos escritos a mano utilizando el conjunto de datos Optical Recognition of Handwritten Digits. Este problema, aparentemente sencillo, no obtuvo un rendimiento perfecto en la clasificación de los dígitos, lo cual detalla la complejidad reflejada en aplicar modelos a problemas que puedan parecer sencillos.

Se exploraron los efectos de cambiar la configuración de la red, incluyendo el número de capas ocultas, el número de neuronas en esas capas y la función de activación. Tal como se pudo observar, al realizar estas configuraciones pudimos obtener un modelo con mejor rendimiento al evaluarlo con la métrica de precisión.

Una vez entrenados los modelos, se aplicó la selección del modelo óptimo mediante la comparación de su precisión en el conjunto de validación y la posterior evaluación mediante una matriz de confusión normalizada.

Este informe busca dar al lector una ejemplificación de aplicación de estrategias en el entrenamiento de modelos de redes neuronales, haciendo uso de gran variedad de distintos conceptos aplicados en el entrenamiento y selección de modelos.

VII. Bibliografía

1) https://www.ibm.com/docs/es/spss-modeler/saas?topic=networks-neural-model
2) https://www.datacentric.es/blog/insight/red-neuronal-artificial-aplicaciones
3) https://datascientest.com/es/perceptron-que-es-y-para-que-sirve
4) https://aiofthings.telefonicatech.com/recursos/datapedia/funcion-activacion
5) https://aws.amazon.com/es/what-is/overfitting/