1st Summer School Physical Engineering UNI 2022-Lima,Perú

Deep Learning

Adrián Israel Arancibia González

Universidad de Santiago de Chile

adrian.arancibiag@usach.cl

February 24, 2022

Resumen

En el siguiente informe se presentan los resultados del entrenamiento de un modelo de red neuronal artificial entrenada para identificar los 26 signos alfabéticos del American Sign Lenguage (ASL), "delete", "espacio" y la ausencia de signo, en total 29 salidas, a partir del dataset ASL de Akash en Kaggle.

Palabras Clave: Deep Learning, Multilayer Artificial Neural Network, Multilayer perceptron, ASL, resnet50, PyTorch

1 Metodología

Las redes neuronales se componen en general de un modelo que define los parámetros y métodos que ocupa la red neuronal, un criterio de perdida, que nos define que tan lejos esta el estado actual del modelo de predecir los resultados esperados y un método de optimización, el cual permite reducir, bajo el entrenamiento de la red, la función de perdida del modelo[1].

El dataset ASL de Akash en Kaggle ([2]) contiene 87000 imágenes de manos realizando signos del ASL detallando cual es el valor de dicho signo (de la A a la Z sin ñ más el signo de espacio y la ausencia de signo), estas imágenes se separaron en forma aleatoria en 69600 imágenes para entrenamiento y 17400 imágenes para testear el modelo, las cuales se subdividieron en grupos de 32 imágenes (batch size).

A lo largo del proyecto se trabajo con el lenguaje de programación Python en especial utilizando la librería PyTorch. Se intento partir definiendo una red multilayer con todos los parámetros libres pero dado el numero de salidas y el número de parámetros a entrenar, el proceso de optimización resulto muy lento por lo que se decidió aplicar un modelo preentrenado para distinguir imágenes. El modelo elegido fue resnet50, modelo de 50 layers preentrenado para identificar 1000 imágenes, los parámetros internos se mantuvieron fijos dejando solo un total de 59421 parámetros entrenables y 23508032 parámetros fijos, con 244x244x3 parámetros de entrada (imagen RGB de 244x244) y 29 clases de salida 28 símbolos y una clase para fotos sin signos [3]

```
model = torchvision.models.resnet50(pretrained=True)


for param in model.parameters():
    param.requires_grad = False

in_features = model.fc.in_features
model.fc = torch.nn.Linear(in_features, num_classes)
```

se utilizó una función de perdida Cross Entropy Loss, y un optimizador Adam con una razón de aprendizaje de 0.001.

```
crit = nn.CrossEntropyLoss()
optim = torch.optim.Adam(model.parameters(),lr=learning_rate)
```

La función de entrenamiento de la red "train()" itera sobre las épocas y sobre el dataset de entrenamiento, cada época evalúa el modelo, calcula el criterio de distancia entre los valores predichos y los reales a través de la "loss function" y optimiza los parámetros utilizando el gradiente de la "Loss function" con respecto al la salida de la red, luego evalúa el modelo sobre el data set de pruebas y calcula la exactitud del modelo, para obtener información de cada época esta función puede imprimir la numero de la época, los valores de la perdida (Loss function evaluada) para los datases de entrenamiento y de prueba, la exactitud de los resultados de entrenamiento y de prueba, el tiempo de ejecución de cada época y el tiempo restante aproximado, entre otros.

2 Resultados

Se entrenaron 10 épocas y se guardo el estado completo del modelo y el optimizador usando la función “torch.save” en un archivo “.ckpt”:

```
torch.save({
            'epoch': 28,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optim.state_dict(),
            'train_loss': 0.04705794344725751
}, "train.ckpt")
```

en este tramo desde un porcentaje de éxito del 91,56 % se alcanzó un 97.59 %.

Luego se cargo el estado completo del modelo y el optimizador unsando la función “torch.load”:

```
checkpoint = torch.load("train.ckpt")
model.load_state_dict(checkpoint['model_state_dict'])
optim.load_state_dict(checkpoint['optimizer_state_dict'])
epoch = checkpoint['epoch']
loss = checkpoint['train_loss']
```

y se volvieron a entrenar 10 épocas obteniendo un porcentaje de éxito de 98,28 %.

```
train(model,train_dataloader, test_dataloader, crit, optim, num_epoch)
✓ 233m 8.9s
epoch: 0, train loss: 0.7507062224204513, train acc: 81.5905172413792%, test loss: 0.3235555437721885, test acc: 91.5574712643678%
epoch: 1, train loss: 0.2833136107164553, train acc: 92.0580004597%, test loss: 0.2001500242603006, test acc: 91.13793103448276%
epoch: 2, train loss: 0.2113237763324088Z, train acc: 93.67672413793103%, test loss: 0.1781962963116410Z, test acc: 94.71264367816092%
epoch: 3, train loss: 0.1727808702599193Z, train acc: 94.64808459770115%, test loss: 0.1208093242466395, test acc: 96.51149425287356%
epoch: 4, train loss: 0.1478726178818735Z, train acc: 95.41091954022988%, test loss: 0.1014570126049768S, test acc: 97.10919540229885%
epoch: 5, train loss: 0.1349810237558042Z, train acc: 95.66379310344827%, test loss: 0.0931508799044155, test acc: 97.22988505747126%
epoch: 6, train loss: 0.1192976006567006Z, train acc: 96.28735632183908%, test loss: 0.0912364595295751Z, test acc: 96.97126436781609%
epoch: 7, train loss: 0.1070943621342637Z, train acc: 96.56465517241386%, test loss: 0.0951626116079483Z, test acc: 96.81609195402306%
epoch: 8, train loss: 0.1043745425822020Z, train acc: 96.67385057471265%, test loss: 0.0798907230654389, test acc: 97.56321839080046%
epoch: 9, train loss: 0.0949679873260703S, train acc: 96.84482758620689%, test loss: 0.0758208832994569S, test acc: 97.59195402298857%

train(model,train_dataloader, test_dataloader, crit, optim, num_epoch)
✓ 232m 58.9s
epoch: 0, train loss: 0.0881574926202014S, train acc: 97.11637931034483%, test loss: 0.0679227073675192, test acc: 97.81034482758626%
epoch: 1, train loss: 0.0833136667669804S, train acc: 97.28448275862068%, test loss: 0.0916971699035028, test acc: 96.80655172413794%
epoch: 2, train loss: 0.0790404231931971Z, train acc: 97.43247126436782%, test loss: 0.0750037018618005Z, test acc: 97.45977011494253%
epoch: 3, train loss: 0.0766454471952827S, train acc: 97.42385057471265%, test loss: 0.0797595037759439Z, test acc: 97.27586206896552%
epoch: 4, train loss: 0.0703945057744766, train acc: 97.68821839080463%, test loss: 0.0613111750833067, test acc: 98.06321839080046%
epoch: 5, train loss: 0.0683585055423896, train acc: 97.77298850574712%, test loss: 0.0683015187965973I, test acc: 98.08574712643679%
epoch: 6, train loss: 0.0655580100183484S, train acc: 97.8800804597%, test loss: 0.0585899002297944, test acc: 97.98275862068965%
epoch: 7, train loss: 0.0617872489301016S, train acc: 97.93103448275862%, test loss: 0.0781375856810936, test acc: 97.36206896551724%
epoch: 8, train loss: 0.0600156703931043, train acc: 97.96982758620689%, test loss: 0.0583945191084404Z, test acc: 98.03448275862068%
epoch: 9, train loss: 0.0587595549465043, train acc: 98.09770114942529%, test loss: 0.0510633458202124Z, test acc: 98.28735632183908%
```

Usando el mismo método se entreno la red durante 6 épocas,usando una la razón de aprendizaje a 0.0005, la mitad de la usada anterior mente, logrando llegar en la época 24 a un máximo de éxito del 98.489 %, siendo almacenada las 4ta y la 6ta épocas con porcentaje de éxito de 98,425 % y 98,442 % respectivamente.

```
train(model,train_dataloader, test_dataloader, crit, optim, 5)

epoch: 0, train loss: 0.0551645810869468S4, train acc: 98.16091954022988%, test loss: 0.0733254889160247S, test acc: 97.46551724137932%
epoch: 1, train loss: 0.0567778144266467I, train acc: 98.11063218390805%, test loss: 0.0480601280841630G, test acc: 98.48850574712644%
epoch: 2, train loss: 0.0527117911570847444, train acc: 98.19540229885058%, test loss: 0.0489306613203250O7, test acc: 98.43103448275862%
epoch: 3, train loss: 0.0522040521115E0946, train acc: 98.26293103448276%, test loss: 0.0462001154200816G, test acc: 98.48850574712644%
epoch: 4, train loss: 0.0483813628504031l4, train acc: 98.36925287356321%, test loss: 0.0475755890356403A, test acc: 98.42528735632183%

train(model,train_dataloader, test_dataloader, crit, optim, 2)

epoch: 0, train loss: 0.0488074772581779Z5, train acc: 98.39511494252874%, test loss: 0.0618566812547000B, test acc: 97.79885057471265%
epoch: 1, train loss: 0.0459256641075400A4, train acc: 98.46264367816092%, test loss: 0.0462457860483040B4, test acc: 98.44252873563218%
```

Los resultados son bastante exactos, con una tasa de 3 errores cada 200 predicciones.

En la Fig. 1 podemos ver los resultados de prueba para las 29 clases los cuales muestran una dispersión prácticamente nula reafirmando la calidad del modelo.

3 Discusión de resultados

En comparación con los primeros acercamientos utilizando multilayer lineales convolucionales con todos sus parámetros libres, la utilización el modelo resnet50 (50 layers) preentrenado para distinguir mil imágenes permite mejorar la rapidez en la búsqueda de exactitud en el modelo, las capas internas que no se modificaron contienen una capacidad tremenda de catalogar la información dentro de las imágenes siendo evidente su robustez en la excelente capacidad de predicción del modelo obtenido.

Conclusiones

Este tipo de tecnologías esta cada vez más presente en nuestra realidad, debido a su cada vez más simple aplicación a diversos fenómenos perceptivos, ya sea selección de objetos en imágenes, definición de número de objetos, posiciones, letras, sonidos etc..., además de su utilidad en toma de decisiones.

Aunque estás ya son tecnologías existentes es interesante pensar en utilizar los resultados aquí obtenidos para el desarrollo de traductores que permitan avances en la integración de personas con habilidades diferentes, complementado los resultados con un reconocimiento de voz y un traductor gráfico a lengua de señas.

Agradecimientos

El autor agradece a la Asociación de Estudiantes de Pregrado de Ingeniería Física (AEPIF) de la Universidad Nacional de Ingeniería del Perú por la hospitalidad y la financiación del curso, al Departamento de Física de la Universidad de Santiago de Chile por la financiación parcial de la asistencia al curso y el presente informe de resultados.

Referencias

[1] Elvin Mark Munoz Vega, Curso de Deep Learning, 1st Summer School Physical Engineering UNI 2022-Lima,Perú

[2] https://www.kaggle.com/grassknoted/asl-alphabet

[3] https://www.kaggle.com/julichitai/asl-alphabet-classification-using-pytorch

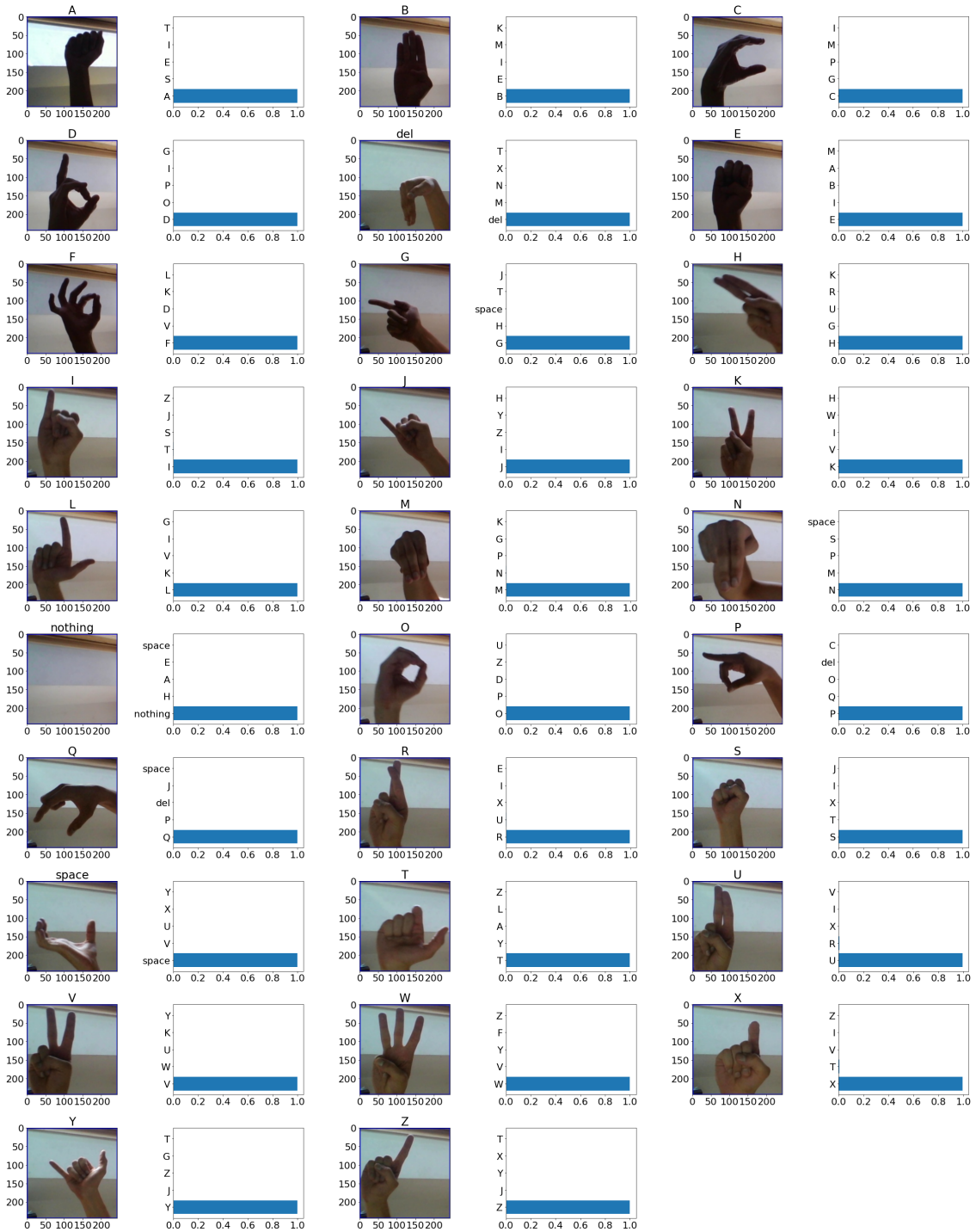

Figura 1: Resultados de prueba, la imagen de entrada es de tamaño 244x244 pixeles, como salidas se muestran las 5 clases más probables, siendo notoria la baja dispersión en los resultados del modelo y la alta precisión de las predicciones .