Instituto Tecnológico y de Estudios Superiores de Monterrey Campus Querétaro

CONCENTRACIÓN DE CIENCIA DE DATOS E INTELIGENCIA ARTIFICIAL AVANZADA

Carlos Eduardo Ortega Clement A01707480

DEEP LEARNING IMPLEMENTATION

Sign Language Alphabet

Este problema busca poder interpretar a texto el alfabeto del lenguaje de señas, para poder realizar esto se encontró un dataset con varios ejemplos de cada una de las palabras del abecedario en español.

En este documento explicaré los cambios realizados en el código para mejorar el modelo. El código completo estará explicado en el mismo archivo de python.

Para mi primera versión del modelo realicé una red neuronal convolutiva no tan compleja:

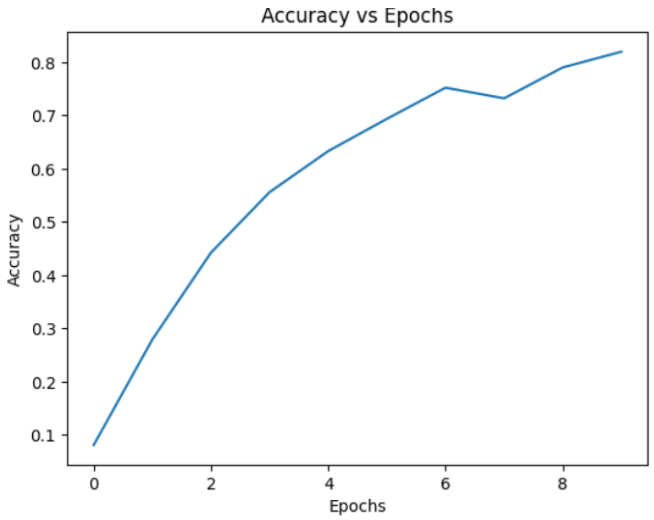
```python
# Build the Sequential convolutional neural network model

def get_model(input_shape):

    model  =  Sequential([
                    Conv2D(32, kernel_size=3, padding='same', activation="relu", input_shape=input_shape),
                    MaxPooling2D((2,2)),
                    #Flatten(),
                    Conv2D(32, kernel_size=3, padding='same', strides = 1, activation="relu"),
                    Conv2D(16, kernel_size=3, padding='same', strides = 1, activation="relu"),
                    Conv2D(16, kernel_size=3, padding='same', strides = 1, activation="relu"),
                    MaxPooling2D((2,2)),
                    Flatten(),
                    Dense(29, activation='softmax')
    ])
    return model
```

Para todas las capas mantuve el mismo padding, activación relu y solamente realicé dos MaxPooling para reducir las dimensiones. De igual manera, para la compilación del modelo utilicé como optimizador “Adam” y de loss “categorical Cossentropy”. Obteniendo como resultado 0.76 de accuracy en el set de validación.

```
Epoch 10/10
100/100 [==============================] - 27s 265ms/step - loss: 0.6340 - accuracy: 0.7970 - val_loss: 0.7975 - val_accur
acy: 0.7650
```

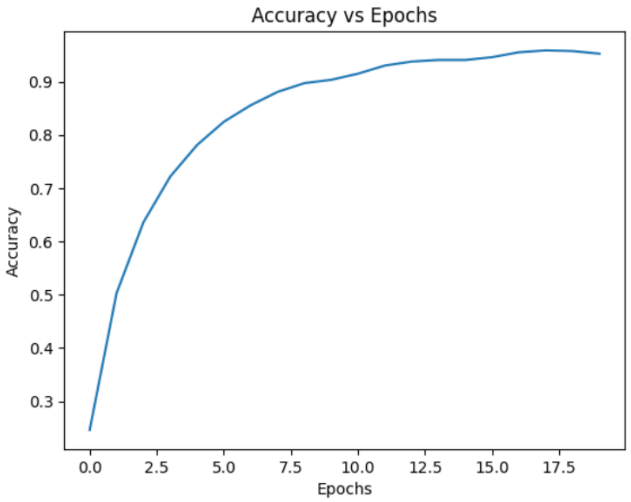


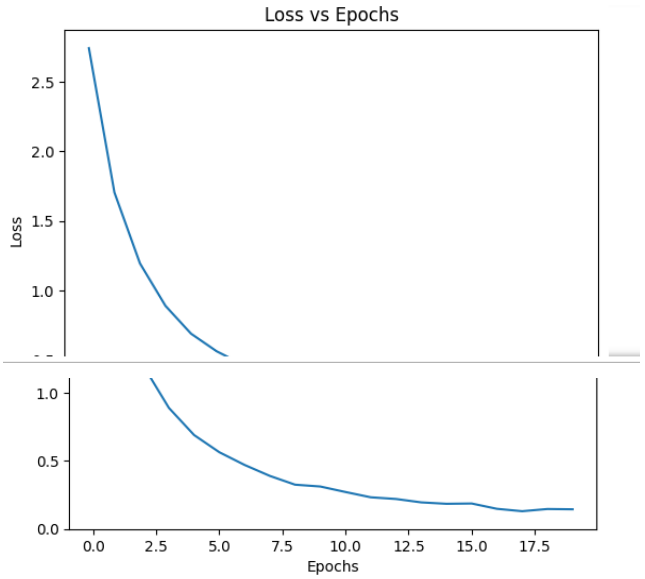
MEJORAS EN EL MODELO:

El primer ajuste que realicé fue en los hiper parámetros como aumentar el número de epochs, el tamaño del step_per_epoch y del validation step. En este caso obtuve un accuracy mucho más alto:

```python
steps_per_epoch =200,
epochs=20,  # Número de épocas de entrenamiento
validation_data=val_generator,
validation_steps = 100
```

```
Epoch 20/20
200/200 [==============================] - 50s 248ms/step - loss: 0.1446 - accuracy: 0.9525 - val_loss: 0.1508 - val_accur
acy: 0.9520
```





Pero después de esto, noté que existía un poco de sobre ajuste en el modelo por lo que ahora modifiqué el modelo agregando otra capa densa y un drop out para evitar este sobreajuste, quedando de la siguiente manera:

```
model  =  Sequential([
                Conv2D(32, kernel_size=3, padding='same', activation="relu", input_shape=input_shape),
                MaxPooling2D((2,2)),
                #Flatten(),
                Conv2D(32, kernel_size=3, padding='same', strides = 1, activation="relu"),
                Conv2D(16, kernel_size=3, padding='same', strides = 1, activation="relu"),
                MaxPooling2D((2, 2)),
                Conv2D(16, kernel_size=3, padding='same', strides = 1, activation="relu"),
                MaxPooling2D((2,2)),
                Flatten(),
                Dropout(0.5),
                Dense(128, activation='relu'),
                Dense(29, activation='softmax')
])
```

De igual forma modifiqué parte del image data generator para que sea más robusto:

Agregué un zoom range, un width y height shift range para poder tener más variabilidad en las imágenes. No pude transformar de otras formas las imágenes ya que para este problema si es importante mantener la rotación de las imágenes.

```
# Generador de flujo de datos para el conjunto de entrenamiento
train_datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=False,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    validation_split=0.2  # Especifica el porcentaje de datos para validación
)
```

Al realizar esto, obtuve un accuracy demasiado bajo,aunque modificara mucho mi modelo con más capas y dropouts:

```
Epoch 18/20
200/200 [==============================] - 56s 279ms/step - loss: 1.0459 - accuracy: 0.6768 - val_loss: 1.6026 - val_accur
acy: 0.5410
Epoch 19/20
200/200 [==============================] - 56s 278ms/step - loss: 1.0413 - accuracy: 0.6793 - val_loss: 1.5764 - val_accur
acy: 0.5025
Epoch 20/20
124/200 [=================>............] - ETA: 18s - loss: 0.9808 - accuracy: 0.6944
```

Sin embargo, no decidí volver al modelo que me dió el accuracy de .95 ya que las imágenes no tenían variación y eso afectaba demasiado a imágenes nuevas, diferentes a las de entrenamiento. Por esta razón decidí utilizar transfer learning con la arquitectura VGG16.

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))


# Congelar las capas convolucionales del modelo base
for layer in base_model.layers:
    layer.trainable = False

# Agregar capas personalizadas para la clasificación de señas
x = Flatten()(base_model.output)
output = Dense(29, activation='softmax')(x)

# Crear el modelo completo
model = Model(inputs=base_model.input, outputs=output)

model.summary()

model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model.fit(
        train_generator,
        steps_per_epoch =200,
        epochs=20,  # Número de épocas de entrenamiento
        validation_data=val_generator,
        validation_steps = 100
    )
```

Lo que hice fue utilizar la arquitectura VGG16 ya que tiene capas más complejas que las que yo utilicé haciendo el modelo más robusto. Congele las capas anteriores y agregué una capa densa de 29 neuronas(las cuales corresponden a las 29 clases). Mantuve los mismos valores en cuanto a hiper parámetros y entrené mi modelo, dando como resultado lo siguiente:

```
Epoch 13/15
100/100 [==============================] - 92s 929ms/step - loss: 0.1292 - accuracy: 0.9750 - val_loss: 0.1775 - val_accur
acy: 0.9620
Epoch 14/15
100/100 [==============================] - 93s 933ms/step - loss: 0.1184 - accuracy: 0.9755 - val_loss: 0.2467 - val_accur
acy: 0.9230
Epoch 15/15
100/100 [==============================] - 93s 930ms/step - loss: 0.1107 - accuracy: 0.9830 - val_loss: 0.1977 - val_accur
acy: 0.9510
```

Pude notar que en el epoch 13 tuve un mejor accuracy en el set de validación que mis otros modelos. Por lo que para poder quedarme con el accuracy de ese epoch volví a correr el entrenamiento pero esta vez lo frené cuando alcanzó un accuracy similar a 0.96.