DL_2024_2_Trabajo_Corto_1

September 14, 2024

Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Maestría Académica en Ciencias de la Computación

Curso: Electiva Deep Learning

Segundo Semestre 2024

Profesor: Dr. Luis-Alexander Calvo-Valverde

Trabajo Práctico: 1

Datos de la entrega: Jueves 26 de setiembre 2024, a más tardar a las 6:00 pm

Medio de entrega: Por medio del TEC-Digital.

Entregables: Un archivo jupyter (.IPYNB) y todos los archivos adiconales que se requieran para correr su Cuaderno (En un archivo comprimido). En caso de requerir mucho espacio, solicitarle al profesor una carpeta en One-Drive para subir la solución.

Estudiantes: - Estudiante 1 - Estudiante 2

0.1 Leer esto primero.

1. Usted puede cambiar el dataset que se le proporciona por otro que sea de su interés; pero de hacerlo, se le recomienda valorarlo con el profesor para que su dataset propuesto no le agregue una complicación importante al Trabajo Práctico.
2. En caso de que el diseño experimental supere en mucho la capacidad de procesamiento computacional que puede conseguir, se le recomienda hablar con el profesor para valorar opciones como disminuir el tamaño del dataset.

0.2 Indicaciones generales que deben seguir:

1. Se le proporciona el conjunto de datos y una hoja electrónica con detalles del dataset.
2. Realizarán clasificación y el atributo a predecir es: **melanocytic**.
3. Ustedes deben ir tomando las decisiones en el proceso y documentarlas en celdas de texto y además su código debe venir ampliamente comentado.

4. Se dividirá el dataset en tres conjuntos de datos: train (60%), validation (20%) y test (20%).
5. Ustedes proponen el diseño experimental (quiero ver qué han entendido de este concepto fundamental).

1 Parte 1. Experimentación con capas totalmente conectadas y un selector de hiperparámetros

1. Debe proponer una red neuronal artificial que solo incluya capas totalmente conectadas. Para la selección de hiperparámetros debe utilizar una herramienta especializada para esto (como keras tunner).

[17]:
```
# Imports
import os
import pandas as pd
from PIL import Image
import numpy as np
from comet_ml import Experiment
from tensorflow import keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, BatchNormalization
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping
import keras_tuner as kt
import matplotlib.pyplot as plt
```

[9]:
```
# Preprocess data
file_path = r"metadata.csv"
data = pd.read_csv(file_path)

# verify the data
data.head()

image_folder = 'ISIC-images/'

# Function to load and preprocess an image
def load_image(image_id, target_size=(128, 128)):
    image_path = os.path.join(image_folder, f'{image_id}.jpg')
    img = Image.open(image_path).resize(target_size)
    # Normalize the image
    img = np.array(img) / 255.0
    return img

# Apply the function to all image IDs in the CSV
image_data = []
for image_id in data['isic_id']:
    try:
```

```
        img = load_image(image_id)
        image_data.append(img)
    except FileNotFoundError:
        print(f"Image {image_id} not found")

# Convert to a NumPy array
X_images = np.array(image_data)

# Labels are in the 'melanocytic' column
y_labels = data['melanocytic'].values

# Split the dataset into training, validation, and test sets
from sklearn.model_selection import train_test_split

X_train, X_temp, y_train, y_temp = train_test_split(X_images, y_labels,␣
↪test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,␣
↪random_state=42)

X_train.shape, X_val.shape, X_test.shape
```

[9]:
```
        isic_id                                        attribution  \
0  ISIC_0024306  ViDIR Group, Department of Dermatology, Medica…
1  ISIC_0024307  ViDIR Group, Department of Dermatology, Medica…
2  ISIC_0024308  ViDIR Group, Department of Dermatology, Medica…
3  ISIC_0024309  ViDIR Group, Department of Dermatology, Medica…
4  ISIC_0024310  ViDIR Group, Department of Dermatology, Medica…

  copyright_license  age_approx anatom_site_general benign_malignant  \
0          CC-BY-NC        45.0                 NaN           benign
1          CC-BY-NC        50.0     lower extremity           benign
2          CC-BY-NC        55.0                 NaN           benign
3          CC-BY-NC        40.0                 NaN           benign
4          CC-BY-NC        60.0      anterior torso        malignant

   concomitant_biopsy diagnosis            diagnosis_confirm_type  \
0               False     nevus  serial imaging showing no change
1               False     nevus  serial imaging showing no change
2               False     nevus  serial imaging showing no change
3               False     nevus  serial imaging showing no change
4                True  melanoma                    histopathology

    image_type   lesion_id  melanocytic     sex
0  dermoscopic  IL_7252831         True    male
1  dermoscopic  IL_6125741         True    male
2  dermoscopic  IL_3692653         True  female
3  dermoscopic  IL_0959663         True    male
```

```
4  dermoscopic  IL_8194852        True    male
```

[4]:
```
# Model with fully connected layers
def build_fc_model():
    model = Sequential()

    # Flatten the image input (128x128x3) into a 1D vector
    model.add(Flatten(input_shape=(128, 128, 3)))

    # Add a few fully connected (dense) layers
    model.add(Dense(512, activation='relu'))  # First dense layer
    model.add(Dense(256, activation='relu'))  # Second dense layer
    model.add(Dense(128, activation='relu'))  # Third dense layer

    # Output layer for binary classification
    model.add(Dense(1, activation='sigmoid'))  # Sigmoid for binary␣
 ↪classification

    # Compile the model
    model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

    return model
```

```
2024-09-10 22:26:50.646936: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:485] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-09-10 22:26:50.718177: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:8454] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-09-10 22:26:50.740467: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1452] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2024-09-10 22:26:50.850349: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
2024-09-10 22:26:52.262590: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

[5]:
```
# hyperparameter tuning for Fully connected layers
def build_tuner_fc_model(hp):
```

```
    model = Sequential()

    # Flatten the image input (128x128x3) into a 1D vector
    model.add(Flatten(input_shape=(128, 128, 3)))

    # Tune the number of dense layers (1 to 4)
    for i in range(hp.Int('num_dense_layers', 1, 4)):
        # Tune the number of units in each Dense layer (between 32 and 512)
        hp_units = hp.Int(f'units_{i}', min_value=32, max_value=512, step=32)
        model.add(Dense(units=hp_units, activation='relu'))

    # binary classification
    model.add(Dense(1, activation='sigmoid'))

    # Tune the learning rate
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
    model.compile(optimizer=keras.optimizers.
  ↪Adam(learning_rate=hp_learning_rate),
                  loss='binary_crossentropy', metrics=['accuracy'])

    return model
```

```
Trial 28 Complete [00h 00m 26s]
val_accuracy: 0.770477831363678

Best val_accuracy So Far: 0.8263651728630066
Total elapsed time: 02h 47m 43s
```

[6]:
```
## Run the model and the Tuner

# Initialize the tuner
tuner = kt.Hyperband(build_tuner_fc_model,
                     objective='val_accuracy',
                     max_epochs=20,
                     factor=3,
                     directory='tuner_dir',
                     project_name='melanocytic_classification_fine_tuning')

# Perform the hyperparameter search
tuner.search(X_train, y_train, epochs=10, validation_data=(X_val, y_val),␣
  ↪batch_size=32)
# Get the optimal hyperparameters
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]

# Print the best hyperparameters
print(f"Best number of layers: {best_hps.get('num_dense_layers')}")
for i in range(best_hps.get('num_dense_layers')):
```

```python
    print(f"Units in dense layer {i}: {best_hps.get(f'units_{i}')}")
print(f"Best learning rate: {best_hps.get('learning_rate')}")


# Build the final model using the best hyperparameters
final_model = tuner.hypermodel.build(best_hps)

# Train the final model
history = final_model.fit(X_train, y_train, epochs=20, validation_data=(X_val,␣
 ↪y_val), batch_size=32)

# Evaluate the final model on the test set
test_loss, test_accuracy = final_model.evaluate(X_test, y_test)

print(f"Final Test Accuracy: {test_accuracy:.4f}")
```

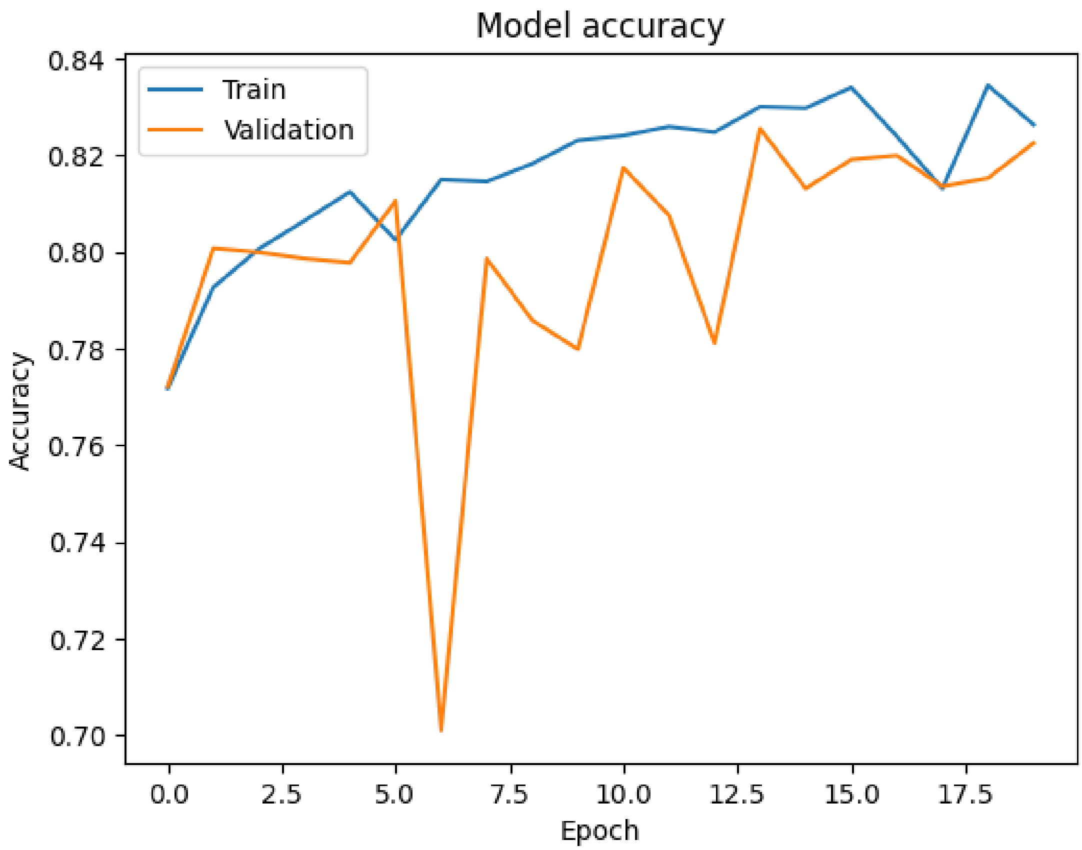
```
Best number of layers: 3
Units in dense layer 0: 288
Units in dense layer 1: 64
Units in dense layer 2: 32
Best learning rate: 0.0001
Epoch 1/20
220/220               5s 14ms/step -
accuracy: 0.7513 - loss: 0.5452 - val_accuracy: 0.7722 - val_loss: 0.5387
Epoch 2/20
220/220               1s 5ms/step -
accuracy: 0.7860 - loss: 0.4541 - val_accuracy: 0.8008 - val_loss: 0.4200
Epoch 3/20
220/220               1s 5ms/step -
accuracy: 0.7995 - loss: 0.4268 - val_accuracy: 0.7999 - val_loss: 0.4247
Epoch 4/20
220/220               1s 5ms/step -
accuracy: 0.8083 - loss: 0.4190 - val_accuracy: 0.7986 - val_loss: 0.4266
Epoch 5/20
220/220               1s 5ms/step -
accuracy: 0.8153 - loss: 0.3939 - val_accuracy: 0.7978 - val_loss: 0.4244
Epoch 6/20
220/220               1s 5ms/step -
accuracy: 0.8027 - loss: 0.4300 - val_accuracy: 0.8106 - val_loss: 0.4194
Epoch 7/20
220/220               1s 5ms/step -
accuracy: 0.8112 - loss: 0.3946 - val_accuracy: 0.7009 - val_loss: 0.5762
Epoch 8/20
220/220               1s 5ms/step -
accuracy: 0.8144 - loss: 0.3989 - val_accuracy: 0.7986 - val_loss: 0.4061
Epoch 9/20
220/220               1s 5ms/step -
```

```
accuracy: 0.8276 - loss: 0.3831 - val_accuracy: 0.7858 - val_loss: 0.4271
Epoch 10/20
220/220              1s 5ms/step -
accuracy: 0.8265 - loss: 0.3808 - val_accuracy: 0.7799 - val_loss: 0.4357
Epoch 11/20
220/220              1s 5ms/step -
accuracy: 0.8352 - loss: 0.3691 - val_accuracy: 0.8174 - val_loss: 0.3976
Epoch 12/20
220/220              1s 5ms/step -
accuracy: 0.8251 - loss: 0.3855 - val_accuracy: 0.8076 - val_loss: 0.4003
Epoch 13/20
220/220              1s 5ms/step -
accuracy: 0.8231 - loss: 0.3897 - val_accuracy: 0.7811 - val_loss: 0.5027
Epoch 14/20
220/220              1s 5ms/step -
accuracy: 0.8410 - loss: 0.3667 - val_accuracy: 0.8255 - val_loss: 0.3907
Epoch 15/20
220/220              1s 5ms/step -
accuracy: 0.8330 - loss: 0.3751 - val_accuracy: 0.8131 - val_loss: 0.3982
Epoch 16/20
220/220              1s 5ms/step -
accuracy: 0.8364 - loss: 0.3607 - val_accuracy: 0.8191 - val_loss: 0.4010
Epoch 17/20
220/220              1s 5ms/step -
accuracy: 0.8307 - loss: 0.3659 - val_accuracy: 0.8200 - val_loss: 0.3925
Epoch 18/20
220/220              1s 5ms/step -
accuracy: 0.8181 - loss: 0.3846 - val_accuracy: 0.8136 - val_loss: 0.3997
Epoch 19/20
220/220              1s 5ms/step -
accuracy: 0.8293 - loss: 0.3728 - val_accuracy: 0.8153 - val_loss: 0.3968
Epoch 20/20
220/220              1s 5ms/step -
accuracy: 0.8216 - loss: 0.3813 - val_accuracy: 0.8225 - val_loss: 0.3918
74/74            0s 3ms/step -
accuracy: 0.8259 - loss: 0.3691
Final Test Accuracy: 0.8200
```

[7]:

```
# Plot training & validation accuracy values for FC
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

```
# Save the trained model
final_model.save('melanocytic_classif_model_fc.keras')
```



2 Parte 2. Experimentación con libertad de escogencia del tipo de capas

1. En esta segunda implementación puede incluir capas tipo CNN y cualquier otra que considere aporta a la solución.
2. Deben utilizar **una de estas herramientas** para dar seguimiento a los resultados en el caso de la red neuronal artificial -En caso de desear utilizar otra herramienta muy similar, solo solicite de previo autorización al profesor-:
   1. https://www.wandb.com/
   2. https://www.comet.ml/site/

[12]:
```
def build_cnn_model():
    model = Sequential()

    # max pooling
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)))
```

```
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # batch normalization
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Third cnn layer
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Flatten the feature maps into 1D
    model.add(Flatten())

    # Fully connected layer with dropout
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))  # Dropout for regularization

    # Output layer for binary classification
    model.add(Dense(1, activation='sigmoid'))

    # Compile the model
    model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])

    return model
```

[ ]:

```
# Function to build the model for hyperparameter tuning
def build_tuner_cnn_model(hp):
    model = Sequential()

    # Tune the number of filters and kernel size for the first convolutional␣
↪layer
    hp_filters = hp.Int('filters_1', min_value=32, max_value=128, step=16)
    hp_kernel_size = hp.Choice('kernel_size_1', values=[3, 5])
    model.add(Conv2D(filters=hp_filters, kernel_size=(hp_kernel_size,␣
↪hp_kernel_size), activation='relu', input_shape=(128, 128, 3)))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Tune the second convolutional layer
    hp_filters_2 = hp.Int('filters_2', min_value=32, max_value=128, step=16)
    hp_kernel_size_2 = hp.Choice('kernel_size_2', values=[3, 5])
    model.add(Conv2D(filters=hp_filters_2, kernel_size=(hp_kernel_size_2,␣
↪hp_kernel_size_2), activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
    # Tune the third convolutional layer
    hp_filters_3 = hp.Int('filters_3', min_value=64, max_value=256, step=32)
    hp_kernel_size_3 = hp.Choice('kernel_size_3', values=[3, 5])
    model.add(Conv2D(filters=hp_filters_3, kernel_size=(hp_kernel_size_3,␣
  ↪hp_kernel_size_3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Flatten the feature maps into 1D
    model.add(Flatten())

    # Tune the number of units in the dense layer
    hp_units = hp.Int('dense_units', min_value=64, max_value=512, step=64)
    model.add(Dense(units=hp_units, activation='relu'))

    # Tune dropout rate
    hp_dropout = hp.Float('dropout_rate', min_value=0.2, max_value=0.5, step=0.
  ↪1)
    model.add(Dropout(hp_dropout))

    # Output layer
    model.add(Dense(1, activation='sigmoid'))

    # Compile the model with a tunable learning rate
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
    model.compile(optimizer=keras.optimizers.
  ↪Adam(learning_rate=hp_learning_rate),
                  loss='binary_crossentropy', metrics=['accuracy'])

    return model
```

[20]:
```
## Run the model and the Tuner

# Initialize the tuner
tuner = kt.Hyperband(build_tuner_cnn_model,
                     objective='val_accuracy',
                     max_epochs=20,
                     factor=3,
                     directory='tuner_dir',
                     project_name='cnn_melanocytic_tuning'
)

# Perform the search for the best hyperparameters
tuner.search(X_train, y_train, epochs=10, validation_data=(X_val, y_val),␣
  ↪batch_size=32)

# Get the best hyperparameters
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
```

```python
# Build the final model with the best hyperparameters
final_model = tuner.hypermodel.build(best_hps)

# Train the final model with the optimal hyperparameters
history = final_model.fit(X_train, y_train, epochs=20, validation_data=(X_val,␣
 ↪y_val), batch_size=32)

# Evaluate the model on the test set
test_loss, test_accuracy = final_model.evaluate(X_test, y_test)
print(f"Final Test Accuracy: {test_accuracy:.4f}")

# Create a Comet experiment
experiment = Experiment(
    api_key="PlpkGnOY9fq5A1vs3IB2p4kZr",
    project_name="melanocytic-classification",
    workspace="chrisarrefall",
    auto_histogram_weight_logging=True,
    auto_histogram_gradient_logging=True,
    auto_histogram_activation_logging=True,
)

# Build the CNN model
model = build_cnn_model()

# Print model summary
print(model.summary())

# Add training parameters
params = {
    'batch_size': 32,
    'epochs': 20,
    'optimizer': 'adam',
    'activation': 'relu',
    'input_shape': (128, 128, 3),
}

# Start training and log metrics with the prefix 'train_'
with experiment.train():
    history = model.fit(X_train, y_train,
                        batch_size=params['batch_size'],
                        epochs=params['epochs'],
                        validation_data=(X_val, y_val),
                        callbacks=[EarlyStopping(monitor='val_loss',␣
 ↪patience=2, min_delta=0.001, restore_best_weights=True)])
```

```python
# Log the test results
with experiment.test():
    loss, accuracy = model.evaluate(X_test, y_test)
    metrics = {
        'loss': loss,
        'accuracy': accuracy
    }
    experiment.log_metrics(metrics)

# Log additional experiment data
experiment.log_parameters(params)
experiment.log_dataset_hash(X_train)  # Creates and logs a hash of your␣
 ↪training data

# End the experiment to ensure all logs are captured
experiment.end()

# Evaluate on test set
test_loss, test_accuracy = model.evaluate(X_test, y_test)

# Log final test accuracy
experiment.log_metric('Test Accuracy', test_accuracy)

print(f"Final Test Accuracy: {test_accuracy:.4f}")
```

```
COMET WARNING: To get all data logged automatically, import
comet_ml before the following modules: keras, tensorflow.
COMET WARNING: As you are running in a Jupyter environment, you
will need to call `experiment.end()` when finished to ensure all metrics and
code are logged before exiting.
COMET INFO: Experiment is live on comet.com
https://www.comet.com/chrisarrefall/melanocytic-
classification/7d712328c41645fb8aac816c289fbf21

/home/chris/.local/lib/python3.9/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_5"

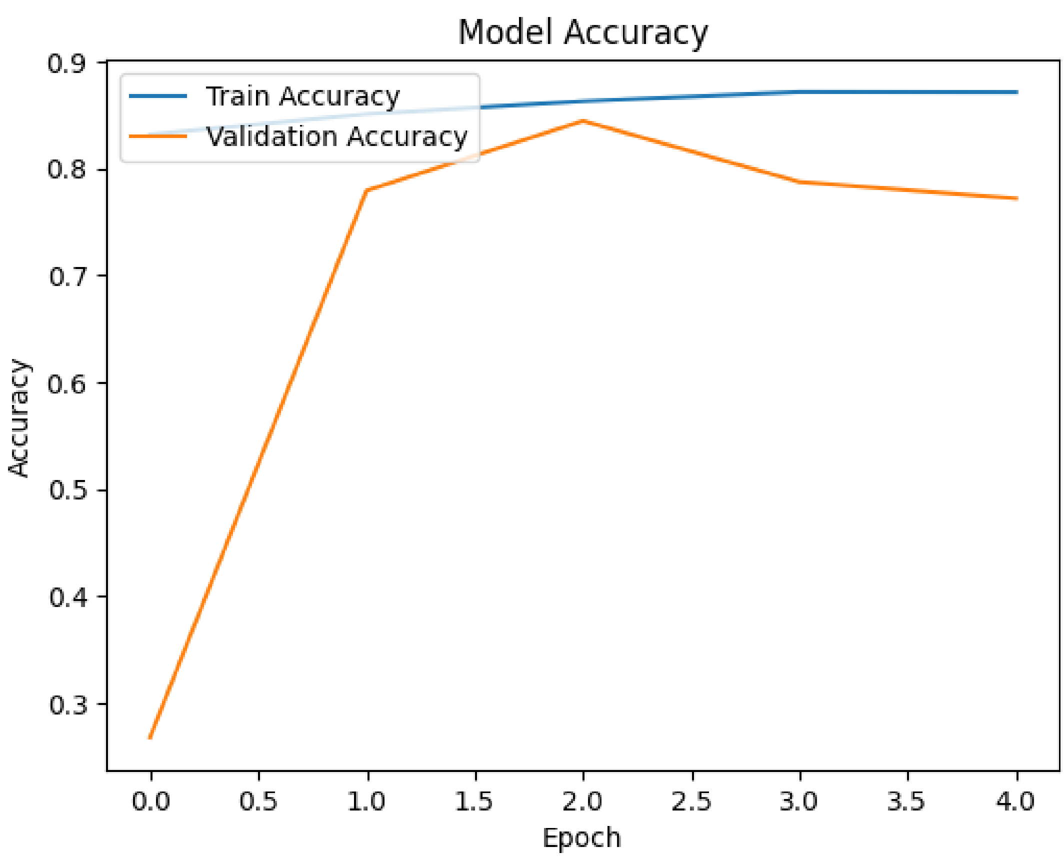
Layer (type)	Output Shape	Param #
conv2d_15 (Conv2D)	(None, 126, 126, 32)	896

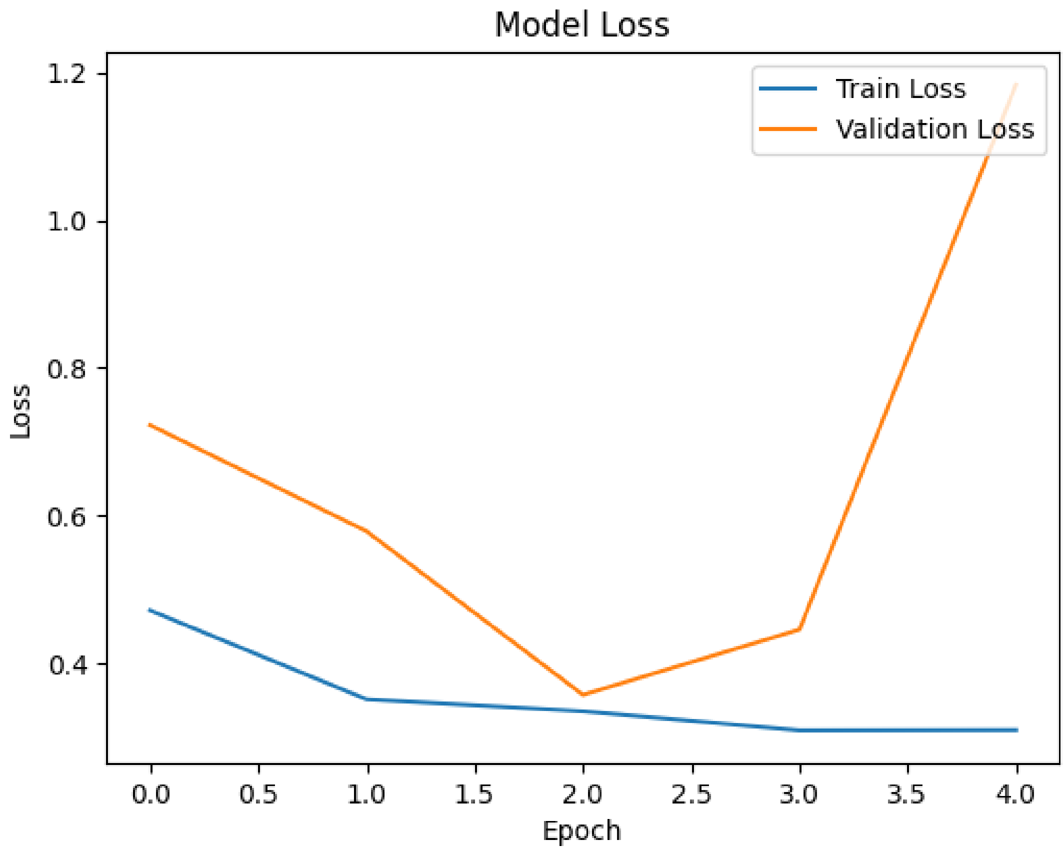
```
 max_pooling2d_15 (MaxPooling2D)   (None, 63, 63, 32)                  0

 conv2d_16 (Conv2D)                (None, 61, 61, 64)             18,496

 batch_normalization_5             (None, 61, 61, 64)                256
 (BatchNormalization)

 max_pooling2d_16 (MaxPooling2D)   (None, 30, 30, 64)                  0

 conv2d_17 (Conv2D)                (None, 28, 28, 128)            73,856

 max_pooling2d_17 (MaxPooling2D)   (None, 14, 14, 128)                 0

 flatten_5 (Flatten)               (None, 25088)                       0

 dense_10 (Dense)                  (None, 512)                12,845,568

 dropout_5 (Dropout)               (None, 512)                         0

 dense_11 (Dense)                  (None, 1)                         513


 Total params: 12,939,585 (49.36 MB)

 Trainable params: 12,939,457 (49.36 MB)

 Non-trainable params: 128 (512.00 B)

None
Epoch 1/20
220/220                   8s 28ms/step -
accuracy: 0.8012 - loss: 0.7833 - val_accuracy: 0.2675 - val_loss: 0.7228
Epoch 2/20
220/220                   2s 9ms/step -
accuracy: 0.8574 - loss: 0.3470 - val_accuracy: 0.7790 - val_loss: 0.5790
Epoch 3/20
220/220                   2s 9ms/step -
accuracy: 0.8635 - loss: 0.3393 - val_accuracy: 0.8443 - val_loss: 0.3576
Epoch 4/20
220/220                   2s 9ms/step -
accuracy: 0.8696 - loss: 0.3121 - val_accuracy: 0.7871 - val_loss: 0.4460
Epoch 5/20
220/220                   2s 9ms/step -
accuracy: 0.8736 - loss: 0.3076 - val_accuracy: 0.7718 - val_loss: 1.1838
74/74                  0s 3ms/step -
```

```
accuracy: 0.8473 - loss: 0.3533
COMET INFO: ----------------------------------------------------
----------------------------------
COMET INFO: Comet.ml Experiment Summary
COMET INFO: ----------------------------------------------------
----------------------------------
COMET INFO:   Data:
COMET INFO:     display_summary_level : 1
COMET INFO:     name                  : sorry_tapir_7424
COMET INFO:     url                   :
https://www.comet.com/chrisarrefall/melanocytic-
classification/7d712328c41645fb8aac816c289fbf21
COMET INFO:   Metrics:
COMET INFO:     test_accuracy : 0.836604118347168
COMET INFO:     test_loss     : 0.3575100302696228
COMET INFO:   Parameters:
COMET INFO:     activation  : relu
COMET INFO:     batch_size  : 32
COMET INFO: Comet.ml Experiment Summary
COMET INFO: ----------------------------------------------------
----------------------------------
COMET INFO:   Data:
COMET INFO:     display_summary_level : 1
COMET INFO:     name                  : sorry_tapir_7424
COMET INFO:     url                   :
https://www.comet.com/chrisarrefall/melanocytic-
classification/7d712328c41645fb8aac816c289fbf21
COMET INFO:   Metrics:
COMET INFO:     test_accuracy : 0.836604118347168
COMET INFO:     test_loss     : 0.3575100302696228
COMET INFO:   Parameters:
COMET INFO:     activation  : relu
COMET INFO:     batch_size  : 32
COMET INFO:     epochs      : 20
COMET INFO:     input_shape : (128, 128, 3)
COMET INFO:     optimizer   : adam
COMET INFO:   Uploads:
COMET INFO:     environment details      : 1
COMET INFO:     filename                 : 1
COMET INFO:     git metadata             : 1
COMET INFO:     git-patch (uncompressed) : 1 (11.95 KB)
COMET INFO:     installed packages       : 1
COMET INFO:     notebook                 : 1
COMET INFO:     os packages              : 1
COMET INFO:     source_code              : 1
COMET INFO:
COMET WARNING: To get all data logged automatically, import
```

comet_ml before the following modules: keras, tensorflow.

[22]:
```
# Plot training & validation accuracy values
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper right')
plt.show()
```





##Criterios de evaluación: 1. Deben presentar una implementación completa para la Parte 1 y para la Parte 2, en una de estas opciones: pytorch, tensorflow o keras (Claro que pueden usar numpy, pandas y otras bibliotecas más, para todo el tema de carga del dataset, analizarlo y pre-procesarlo). (**30 puntos cada una (total 60)**) 1. Uso de herramienta de seguimiento de resultados. (**10 puntos**) 1. Uso de herramienta de selección de hiperparámetros. (**10 puntos**) 1. Documentación de decisiones en celdas de texto y comentarios al código. (**10 puntos**) 1. Conclusiones finales: En una celda de texto al final del cuaderno, incluya sus conclusiones más importantes de los experimentos y algunos de los gráficos que genera la herramienta seleccionada, junto con su interpretación de los mismos. (**10 puntos**)