Deep Learning Medical Image Analysis Example

A simple convolutional neural network model trained using a simple pathology image dataset.

To use GPU acceleration make sure to change your runtime type in Google Colab to GPU.

Python Imports

This section will load the necessary python packages to the instance.

```
# Built-in Imports
import random
```

```
# Library Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
```

```
# Keras Imports
from tensorflow.keras import layers
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.utils import get_file, to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Dataset Downloader

This section will download the selected MedMNIST dataset as a NumPy array object to your Google Colab instance.

To change the dataset that will download just change the variable DATA_NAME to desired dataset name.

All storage on a Google Colab instance is deleted when the instance ends so the dataset will need to be redownloaded each time an instance is created (Don't worry this usually takes about 20 seconds).

RUN FOR BLOODMNIST (MULTI CLASS DATASET)

```
DATA_NAME = "BloodMNIST"
```

RUN FOR BREASTMNIST (BINARY DATASET)

```
DATA_NAME = "BREASTMNIST"
```

RUN FOR BOTH DATASETS

```
!wget https://raw.githubusercontent.com/MedMNIST/MedMNIST/main/medmnist/info.py
from info import INFO
data = INFO[DATA_NAME.lower()]
```

Show hidden output

```
# Downloads the dataset file hosted on Zenodo.
file_path = get_file(fname="dataset.npz",
                     origin=data["url"],
                     md5_hash=data["MD5"])
```

```
A local file was found, but it seems to be incomplete or outdated because the md5 file hash does not match the original valu
Downloading data from https://zenodo.org/records/10519652/files/bloodmnist.npz?download=1
35461855/35461855 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
```

```
# Loads the downloaded NumPy object.
dataset = np.load(file_path)
```

```
# Gets the training images and labels from the NumPy object.
train_x = dataset["train_images"]
train_y = dataset["train_labels"]

# Gets the validation images and labels from the NumPy object.
val_x = dataset["val_images"]
val_y = dataset["val_labels"]

# Gets the testing images and labels from the NumPy object.
test_x = dataset["test_images"]
test_y = dataset["test_labels"]
```

RUN STEP FOR BREASTMNIST

Extra step required for BREASTMNIST since it is gray scale and has the structure (N, 28, 28) and needs the extra channel to match the structure as multi class (N, 28, 28, 1) for example

```
# For BREASTMNIST, check if images are grayscale.
print("Before expand_dims:", train_x.shape)
# If shape is (num_samples, height, width), add channel dimension:
train_x = np.expand_dims(train_x, axis=-1)
val_x   = np.expand_dims(val_x, axis=-1)
test_x  = np.expand_dims(test_x, axis=-1)
print("After expand_dims:", train_x.shape)
```

```
Before expand_dims: (546, 28, 28)
After expand_dims: (546, 28, 28, 1)
```

RUN FOR BOTH DATASETS

Data Exploration

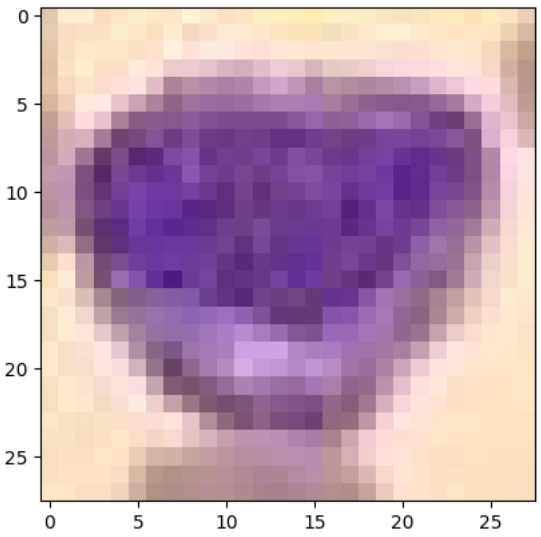
In this section we have a look at our data, their distributions to see if it is ready to be used within our machine learning algorithm.

```
# Declares a list of labels.
labels = list(data["label"].values()) + ["total"]

# Gets the counts for each label in each of our datasets.
_, train_counts = np.unique(train_y, return_counts=True)
_, val_counts = np.unique(val_y, return_counts=True)
_, test_counts = np.unique(test_y, return_counts=True)

# Prints the counts for each label from each dataset.
print(pd.DataFrame(list(zip(np.append(train_counts, [sum(train_counts)]),
                            np.append(val_counts, [sum(val_counts)]),
                            np.append(test_counts, [sum(test_counts)]))),
                   index=labels, columns=["Train", "Val", "Test"]))
```

```
                                                Train   Val  Test
basophil                                          852   122   244
eosinophil                                       2181   312   624
erythroblast                                     1085   155   311
immature granulocytes(myelocytes, metamyelocyte...  2026   290   579
lymphocyte                                        849   122   243
monocyte                                          993   143   284
neutrophil                                       2330   333   666
platelet                                         1643   235   470
total                                           11959  1712  3421
```

```
# Displays a random image from training dataset.
index = random.randint(0, len(train_x))
print(f"{index}: {labels[train_y[index][0]]}")
plt.imshow(train_x[random.randint(0, len(train_x))])
```

```
10840: lymphocyte
<matplotlib.image.AxesImage at 0x7ce944cc4bd0>
```



Data Processing

In this section we will create a data loader for algorithm that will dynamiclly load and augment the data when needed.

```
# Defines the data generator that will be used to augment the images as they are loaded.
data_generator = ImageDataGenerator(featurewise_center=True,
                                    featurewise_std_normalization=True,
                                    horizontal_flip=True,
                                    vertical_flip=True)
```

```
data_generator.fit(np.append(train_x, val_x, 0))
```

Model Definition

In this section we will define the neural network arcitecture.

RUN FOR BLOODMNIST

Only change in this method from method 1 is in the model definition here

```
#       Model Definition Multi Class

# Define the input layer of the model with the size of an image.
input = layers.Input(shape=train_x[0].shape)

# Directly flatten the input image
flatten = layers.Flatten()(input)

# Add dense layers
dense1 = layers.Dense(7, activation="relu")(flatten)
dense2 = layers.Dense(2, activation="relu")(dense1)

# Multi-class final layer:
num_classes = len(np.unique(train_y))  # should be 8 for BloodMNIST
output = layers.Dense(units=num_classes, activation="softmax")(dense2)

# Initilises the defined model and prints summary of the model.
model = Model(inputs=input, outputs=output, name="Model")
model.summary()
```

```
Model: "Model"
```

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 28, 28, 3)	0
flatten_1 (Flatten)	(None, 2352)	0
dense_3 (Dense)	(None, 7)	16,471
dense_4 (Dense)	(None, 2)	16
dense_5 (Dense)	(None, 8)	24

```
Total params: 16,511 (64.50 KB)
Trainable params: 16,511 (64.50 KB)
Non-trainable params: 0 (0.00 B)
```

RUN FOR BREASTMNIST

Model definition for BREASTMNIST - Key change here is replacing the multi class final layer with (Dense(1, activation = "sigmoid"))

```
#       Model Definition (Binary)

# Define the input layer of the model with the size of an image.
input = layers.Input(shape=train_x[0].shape)

# Directly flatten the input image
flatten = layers.Flatten()(input)

# Add dense layers
dense1 = layers.Dense(13, activation="relu")(flatten)
dense2 = layers.Dense(6, activation="relu")(dense1)

# Final output layer: one unit with sigmoid activation for binary classification
output = layers.Dense(units=1, activation="sigmoid")(dense2)

# Initilises the defined model and prints summary of the model.
model = Model(inputs=input, outputs=output, name="Model")
model.summary()
```

Show hidden output

Model Training

This is where we define the training options and then train the model.

RUN FOR BOTH DATASETS

```
# Defines the parameters used during training.
BATCH_SIZE = 64
NUM_EPOCHS = 10
LEARNING_RATE = 0.001
```

Double-click (or enter) to edit

RUN FOR BLOODMNIST

For the Multi Class dataset use loss = "categorical_crossentropy"

```
# Defines the optimiser used to adjust the model weights and compiles the model.
optimiser = SGD(learning_rate=LEARNING_RATE)
model.compile(optimizer=optimiser, loss="categorical_crossentropy", metrics=["accuracy"])
```

RUN FOR BREASTMNIST

For the Binary dataset use loss = "binary_crossentropy"

```
# Defines the optimiser used to adjust the model weights and compiles the model.
optimiser = SGD(learning_rate=LEARNING_RATE)
model.compile(optimizer=optimiser, loss="binary_crossentropy", metrics=["accuracy"])
```

RUN FOR BLOODMNIST

For the multi class dataset

```
# Convert integer labels to one-hot vectors
train_labels = to_categorical(train_y)
val_labels   = to_categorical(val_y)
```

```
# We use the data generator to pass the training and validation data to the model to train it.
history = model.fit(data_generator.flow(train_x, to_categorical(train_y), batch_size=BATCH_SIZE),
                    steps_per_epoch=len(train_x) // BATCH_SIZE,
                    validation_data=data_generator.flow(val_x, to_categorical(val_y), batch_size=BATCH_SIZE),
                    validation_steps=len(val_x) // BATCH_SIZE,
                    epochs=NUM_EPOCHS)
```

Show hidden output

RUN FOR BREASTMNIST

For the Binary Dataset there is no need to convert the labels to_categorical so you can use train_y and val_y directly

```
history = model.fit(data_generator.flow(train_x, train_y, batch_size=BATCH_SIZE),
                    steps_per_epoch=len(train_x) // BATCH_SIZE,
                    validation_data=data_generator.flow(val_x, val_y, batch_size=BATCH_SIZE),
                    validation_steps=len(val_x) // BATCH_SIZE,
                    epochs=NUM_EPOCHS)
```

Show hidden output

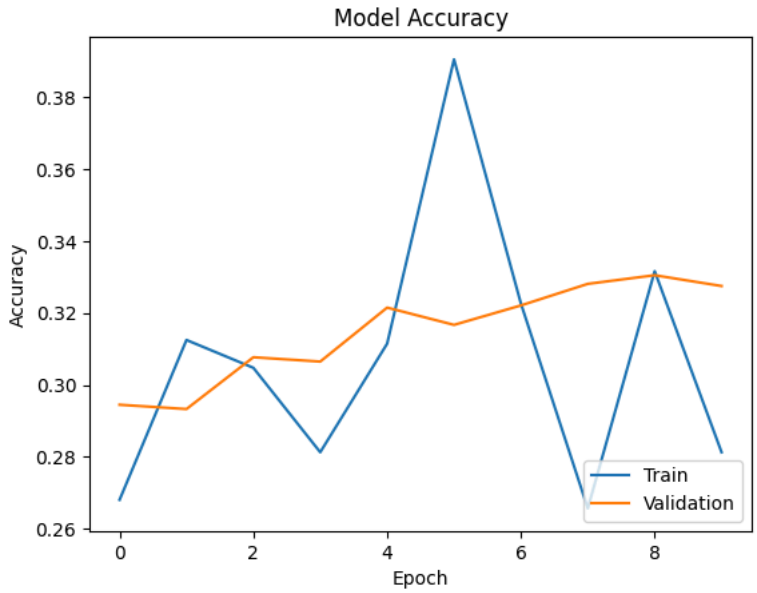
Graphs Below. Duplicated the graph code below for easy comparisons

RUN FOR BLOODMNIST

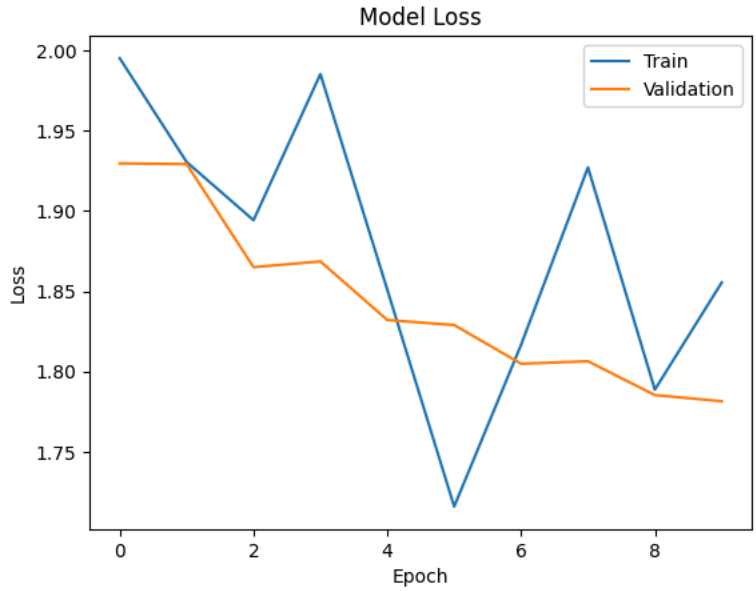
Plot Learning Curves

This is where we visualise the training of the model.

```
# Plots the training and validation accuracy over the number of epochs.
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='lower right')
plt.show()
```




```
# Plots the training and validation loss over the number of epochs.
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```

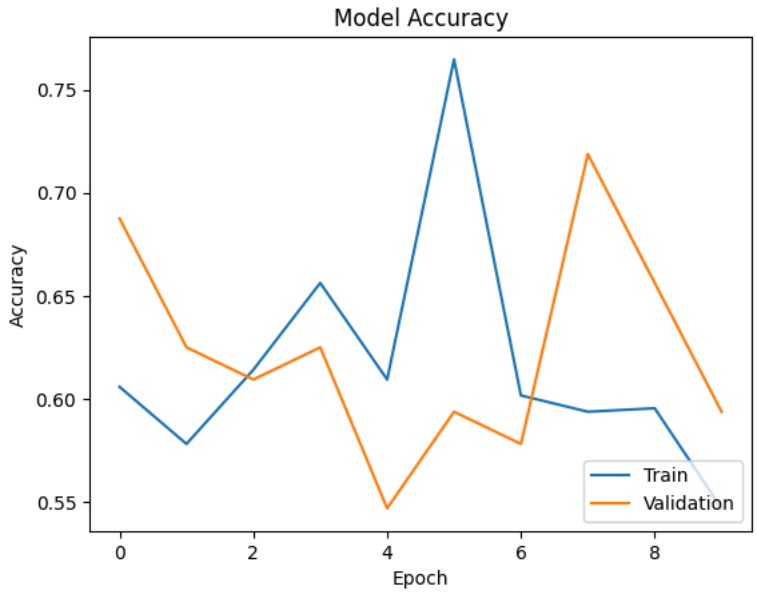


RUN FOR BREASTMNIST

Plot learning curves

This is where we visualise the training of the model.

```
# Plots the training and validation accuracy over the number of epochs.
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='lower right')
plt.show()
```




```
# Plots the training and validation loss over the number of epochs.
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```