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A local file was found, but it seems to be incomplete or outdated because the md5 file hash does not match the original valu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 ━━━━━━━━━━━━━━━━━━━━ 2s 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

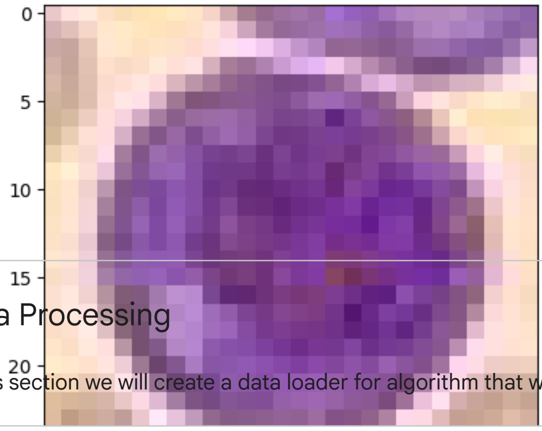
In this section we have a look at our data, their distributions to see if it is ready to be used within our machine learning algorithm.

```
# 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))])
```

```
11841: erythroblast
<matplotlib.image.AxesImage at 0x792c1055dad0>
```



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

```
#       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)

# Defines the first convolutional layer with max pooling.
conv_1 = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(input)
pool_1 = layers.MaxPool2D(pool_size=(2, 2))(conv_1)

# Defines the second convolutional layer with max pooling.
conv_2 = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(pool_1)
pool_2 = layers.MaxPool2D(pool_size=(2, 2))(conv_2)

# Flattens the outputs of the convolutional layers into a one dimensional array.
flatten = layers.Flatten()(pool_2)

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

# 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
conv2d_2 (Conv2D)	(None, 26, 26, 32)	896

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)

# Defines the first convolutional layer with max pooling.
conv_1 = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(input)
pool_1 = layers.MaxPool2D(pool_size=(2, 2))(conv_1)

# Defines the second convolutional layer with max pooling.
conv_2 = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(pool_1)
pool_2 = layers.MaxPool2D(pool_size=(2, 2))(conv_2)

# Flattens the outputs of the convolutional layers into a one dimensional array.
flatten = layers.Flatten()(pool_2)

# Final layer for binary classification: single neuron with sigmoid activation.
output = layers.Dense(1, activation="sigmoid")(flatten)

# 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
```

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

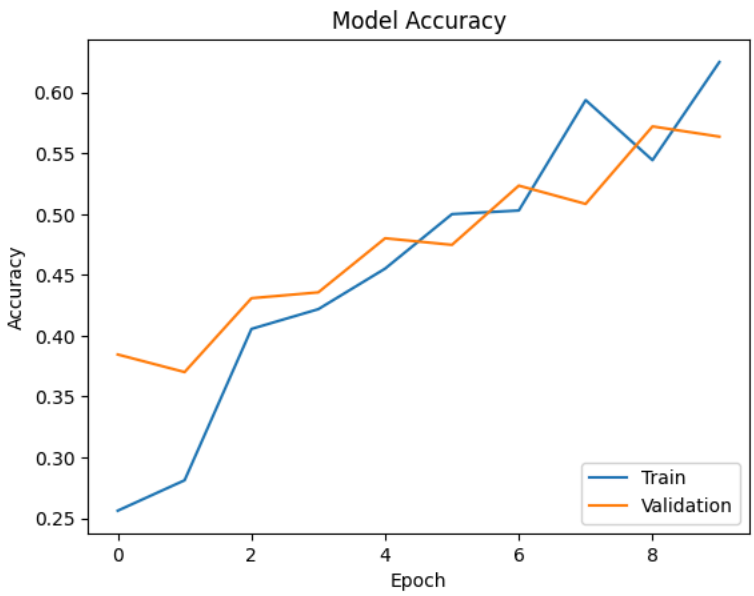
Graphs Below. Duplicated the graph code below for easy comparisons

RUN FOR BLOODMNIST

Plot Learning Curves

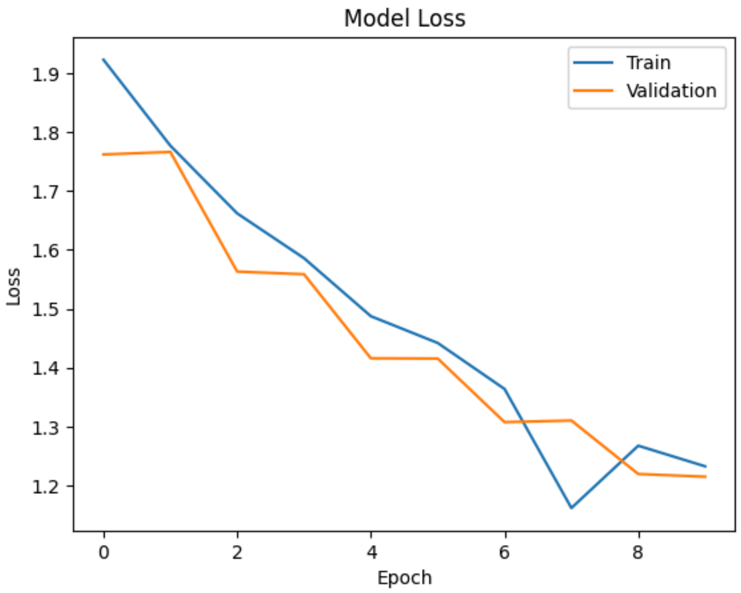
This is where we visualise the training of the model.

```
# 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()
```

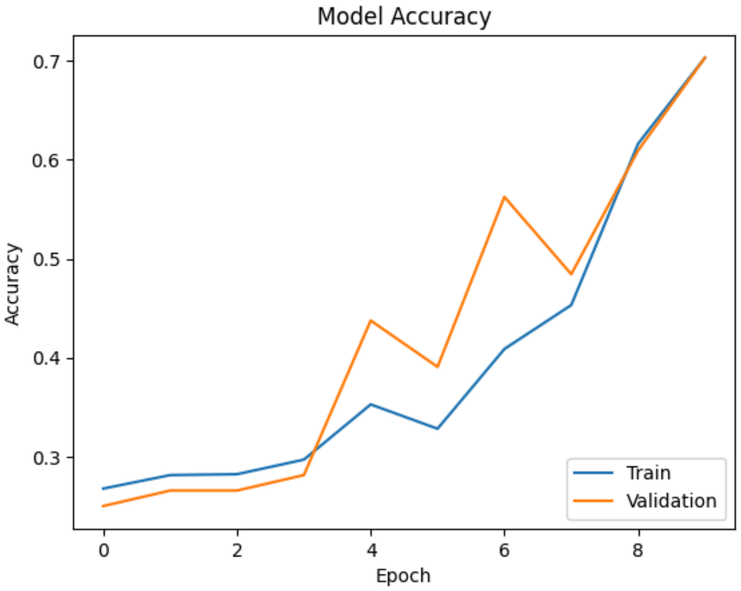


RUN FOR BREASTMNIST

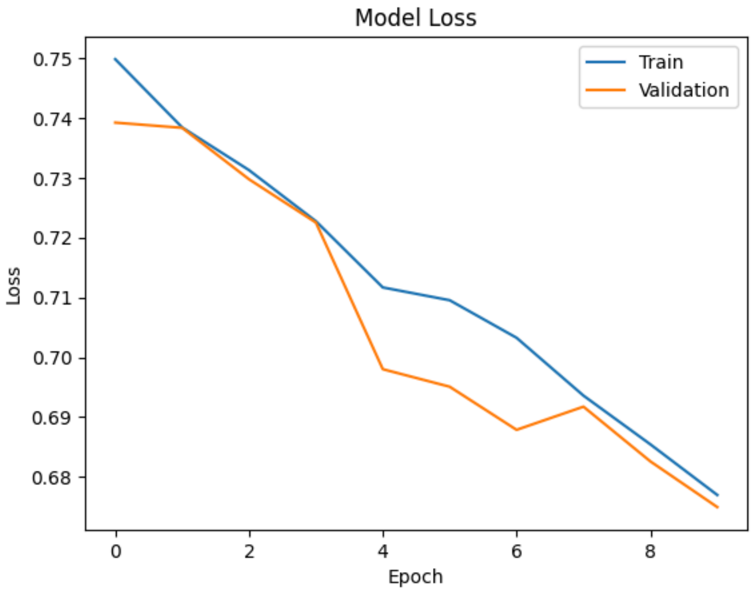
Plot learning curves

This 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()
```



Comparisons of the two datasets for this training method compared to the first method

Method 1

Method 1 uses a simple convolutional neural network, which learns the spatial features through the convolutional layers followed by max-pooling.

Initial observations of Method 1

The convolutions in Method 1 may help retain the spatial relationships, which could be important in medical images like the datasets we are using here.

Method 2

Method 2 skipped the convolutions entirely and only relies on the dense layers. The input images are flattened immediately, which discards the spatial structure.

Initial observations of Method 2

Flattening early on in Method 2 might limit the model's ability to learn patterns, especially for a multiclass dataset like BloodMNIST

Parameter matching

One of the requirements for method 2 was to match as closely as possible the total parameters to Method 1 so that it would be comparable to the convnet used in Method 1.

To match the total parameters, the number of units in the dense layers of Method 2 were reduced by a fair amount, which obviously resulted in a much shallower model.

This reduction, while fair to allow for more comparable results, may have limited Method 2's model's ability to learn the more complex patterns present, especially in the multi class dataset BloodMNIST.

Performance Results

For clarity of the results. Please see above in this notebook for Method 1 showing the individual epochs variables in the training area and the graphs above showing the trends themselves. For Method 2, see the other notebook (Method_2_For_Classification) for the same.

BloodMNIST (Multi Class)

Method 1 performed far better with a higher and more stable validation accuracy. It showed a steady trend upwards over the epochs.

Method 2 had a much lower accuracy and no signs of improvement over the epochs. There was also much more fluctuation in Method 2.

These observations do suggest that the convolutional neural network's ability to capture the spatial features were important.

BreastMNIST (Binary Class)

Method 1 performed similarly with this dataset, showing a steady upward trend in accuracy over the epochs.

Method 2 performed slightly better on this dataset but still contained a lot of fluctuation over the epochs. Its accuracy was slightly higher on this dataset which can be expected on a simpler dataset such as this one.

Conclusion

Both methods followed the same training regime and had fairly similar parameters, Method 1 outperformed Method 2, especially on the multi class dataset. This showcases the importance of convolutional layers in image classification tasks and also highlights how restricting a model's capacity to match the total parameter count of another method can affect its learning ability.