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, Confusi
```

```
# Keras Imports
from tensorflow.keras import layers
from tensorflow.keras.models import Model
from tensorflow import keras
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
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A local file was found, but it seems to be incomplete or outdated because the
Downloading data from https://zenodo.org/records/10519652/files/breastmnist.n
559580/559580 ━━━━━━━━━━━━━━━━━━━━ 0s 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

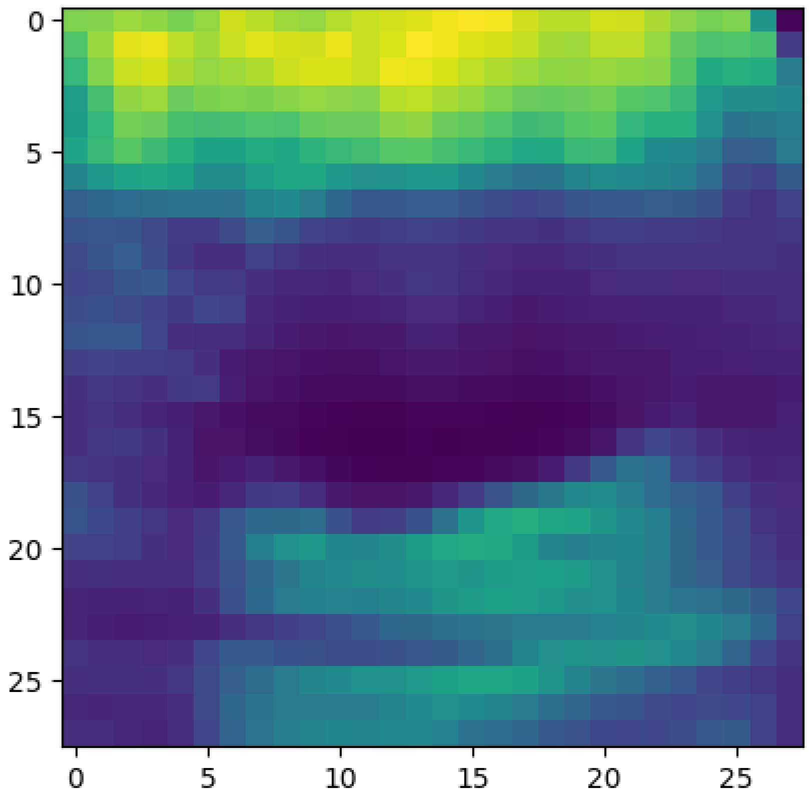
In this section we have a look at our data, their distributions to see if it is ready to be used within our machine learning algorithm.

```
# 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
malignant         147   21    42
normal, benign    399   57   114
total             546   78   156
```

```
# 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))])
```

```
458: normal, benign
<matplotlib.image.AxesImage at 0x78466b635210>
```



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
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

```
#      Method 3 - Custom Deep Network Model Definition (Multi-Class)
input = layers.Input(shape=train_x[0].shape)
```

```
# Conv Block 1
x = layers.Conv2D(32, (3, 3), activation='relu', input_shape=train_x[0].shap
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Dropout(0.25)(x)

# Conv Block 2
x = layers.Conv2D(64, (3, 3), activation='relu')(x)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Dropout(0.25)(x)

# Conv Block 3
x = layers.Conv2D(128, (3, 3), activation='relu')(x)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Dropout(0.25)(x)

# Flatten and Dense Layers
x = layers.Flatten()(x)
x = layers.Dense(128, activation='relu')(x)
x = layers.Dropout(0.5)(x)
num_classes = len(np.unique(train_y))  # e.g., 8 for BloodMNIST
output = layers.Dense(units=num_classes, activation='softmax')(x)

model = Model(inputs=input, outputs=output, name="Model")
model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_c
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "Model"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━
┃ Layer (type)                          ┃ Output Shape                 ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━
│ input_layer_33 (InputLayer)           │ (None, 28, 28, 1)            │
├───────────────────────────────────────┼──────────────────────────────┼───────
│ conv2d_72 (Conv2D)                    │ (None, 26, 26, 32)           │
├───────────────────────────────────────┼──────────────────────────────┼───────
│ max_pooling2d_72 (MaxPooling2D)       │ (None, 13, 13, 32)           │
├───────────────────────────────────────┼──────────────────────────────┼───────
│ dropout_105 (Dropout)                 │ (None, 13, 13, 32)           │
├───────────────────────────────────────┼──────────────────────────────┼───────
│ conv2d_73 (Conv2D)                    │ (None, 11, 11, 64)           │
├───────────────────────────────────────┼──────────────────────────────┼───────
│ max_pooling2d_73 (MaxPooling2D)       │ (None, 5, 5, 64)             │
├───────────────────────────────────────┼──────────────────────────────┼───────
│ dropout_106 (Dropout)                 │ (None, 5, 5, 64)             │
├───────────────────────────────────────┼──────────────────────────────┼───────
│ conv2d_74 (Conv2D)                    │ (None, 3, 3, 128)            │
├───────────────────────────────────────┼──────────────────────────────┼───────
│ max_pooling2d_74 (MaxPooling2D)       │ (None, 1, 1, 128)            │
├───────────────────────────────────────┼──────────────────────────────┼───────
│ dropout_107 (Dropout)                 │ (None, 1, 1, 128)            │
├───────────────────────────────────────┼──────────────────────────────┼───────
│ flatten_33 (Flatten)                  │ (None, 128)                  │
├───────────────────────────────────────┼──────────────────────────────┼───────
│ dense_66 (Dense)                      │ (None, 128)                  │
├───────────────────────────────────────┼──────────────────────────────┼───────
│ dropout_108 (Dropout)                 │ (None, 128)                  │
├───────────────────────────────────────┼──────────────────────────────┼───────
│ dense_67 (Dense)                      │ (None, 2)                    │
└───────────────────────────────────────┴──────────────────────────────┴───────

 Total params: 109,442 (427.51 KB)
 Trainable params: 109,442 (427.51 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"))

More finetuning required compared to BloodMNIST since the dataset was more sensitive.

```
#      Method 3 - Custom Deep Network Model Definition (Binary)
input = layers.Input(shape=train_x[0].shape)

# Conv Block 1
x = layers.Conv2D(16, (3, 3), activation='relu')(input)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Dropout(0.25)(x)
```

```
    # Conv Block 2
    x = layers.Conv2D(32, (3, 3), activation='relu')(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Dropout(0.25)(x)

    # Conv Block 3
    x = layers.Conv2D(64, (3, 3), activation='relu')(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Dropout(0.25)(x)

    # Flatten and Dense Layers
    x = layers.Flatten()(x)
    x = layers.Dense(64, activation='relu')(x)
    x = layers.Dropout(0.5)(x)

    output = layers.Dense(units=1, activation='sigmoid')(x)

    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 BLOODMNIST

```
# Defines the parameters used during training.
BATCH_SIZE = 64
NUM_EPOCHS = 40
LEARNING_RATE = 1e-4
```

RUN FOR BREASTMNIST

Slightly different epochs and learning rate as the dataset was more sensitive.

```
# Defines the parameters used during training.
BATCH_SIZE = 64
NUM_EPOCHS = 30
LEARNING_RATE = 5e-5
```

RUN FOR BLOODMNIST

For the Multi Class dataset use loss = "categorical_crossentropy"

```
# Defines the optimiser used to adjust the model weights and compiles the mo
optimiser = keras.optimizers.RMSprop(learning_rate=LEARNING_RATE)
model.compile(optimizer=optimiser, loss="categorical_crossentropy", metrics=
```

RUN FOR BREASTMNIST

For the Binary dataset use loss = "binary_crossentropy"

```
# Defines the optimiser used to adjust the model weights and compiles the mo
optimiser = keras.optimizers.RMSprop(learning_rate=LEARNING_RATE)
model.compile(optimizer=optimiser, loss="binary_crossentropy", metrics=["acc
```

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
history = model.fit(data_generator.flow(train_x, to_categorical(train_y), ba
                    steps_per_epoch=len(train_x) // BATCH_SIZE,
                    validation_data=data_generator.flow(val_x, to_categorica
                    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_S
                    steps_per_epoch=len(train_x) // BATCH_SIZE,
                    validation_data=data_generator.flow(val_x, val_y, batch_
                    validation_steps=len(val_x) // BATCH_SIZE,
                    epochs=NUM_EPOCHS)
```

Show hidden output

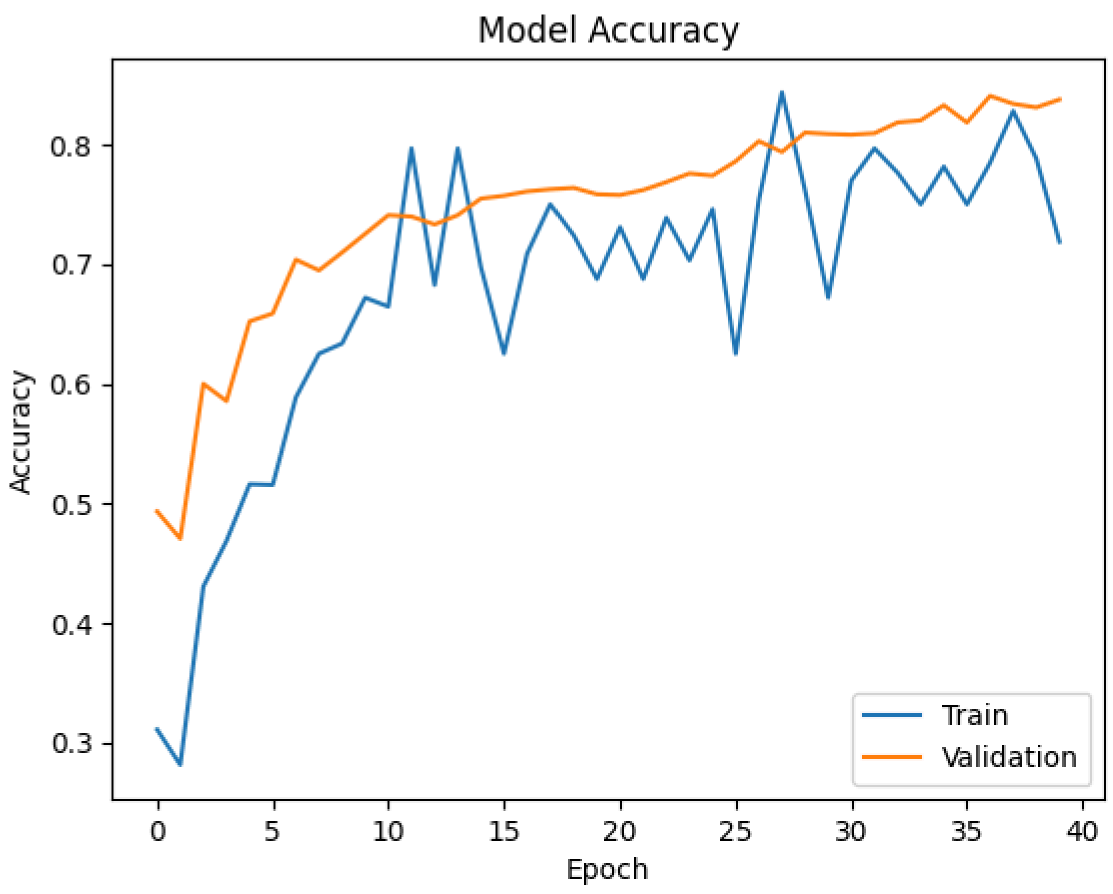
Graphs Below. Duplicated the graph code below for easy comparisons

RUN FOR BLOODMNIST

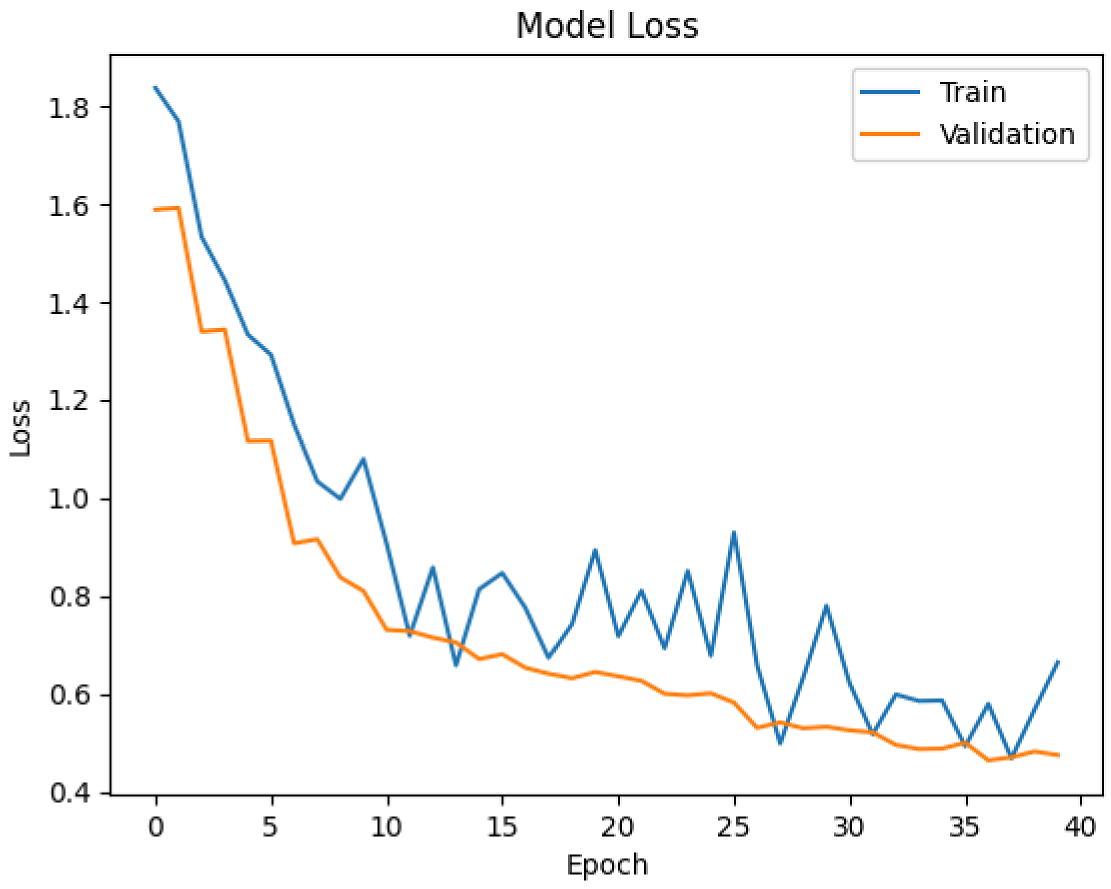
Plot Learning Curves

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()
```

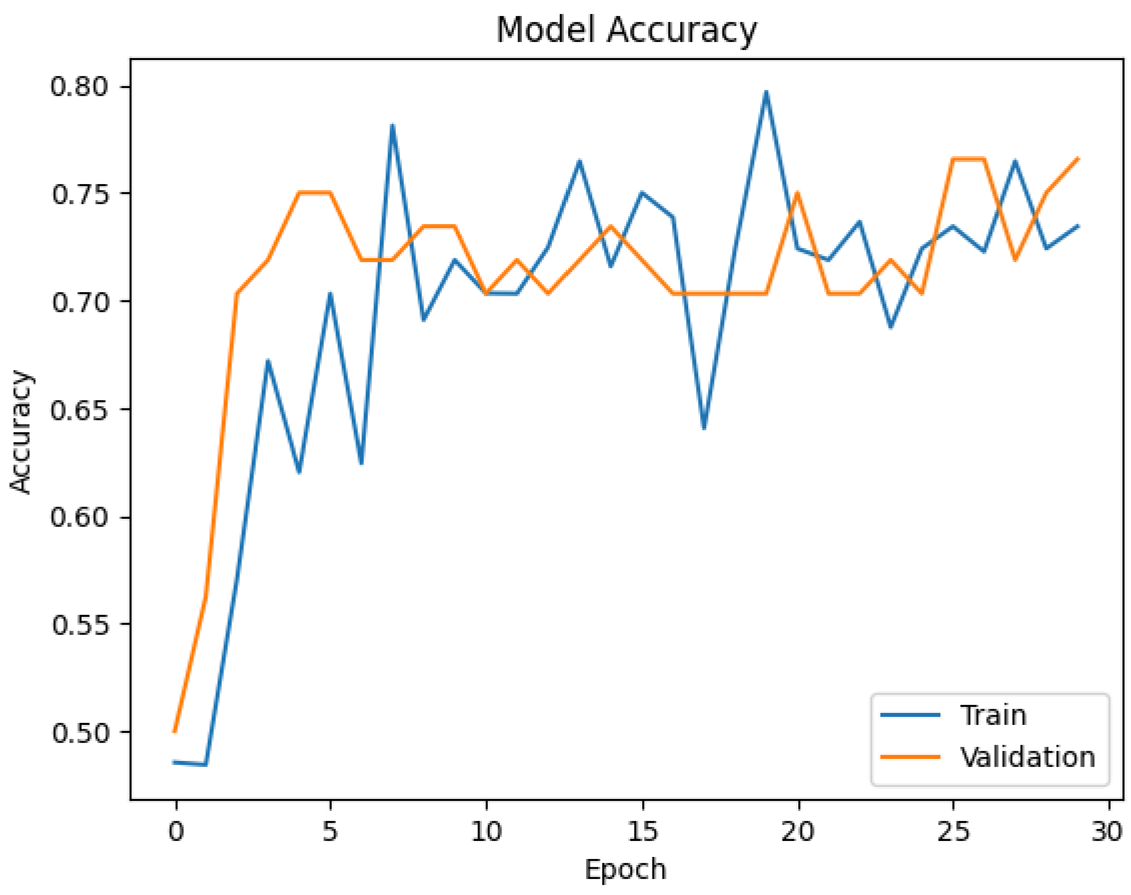


RUN FOR BREASTMNIST

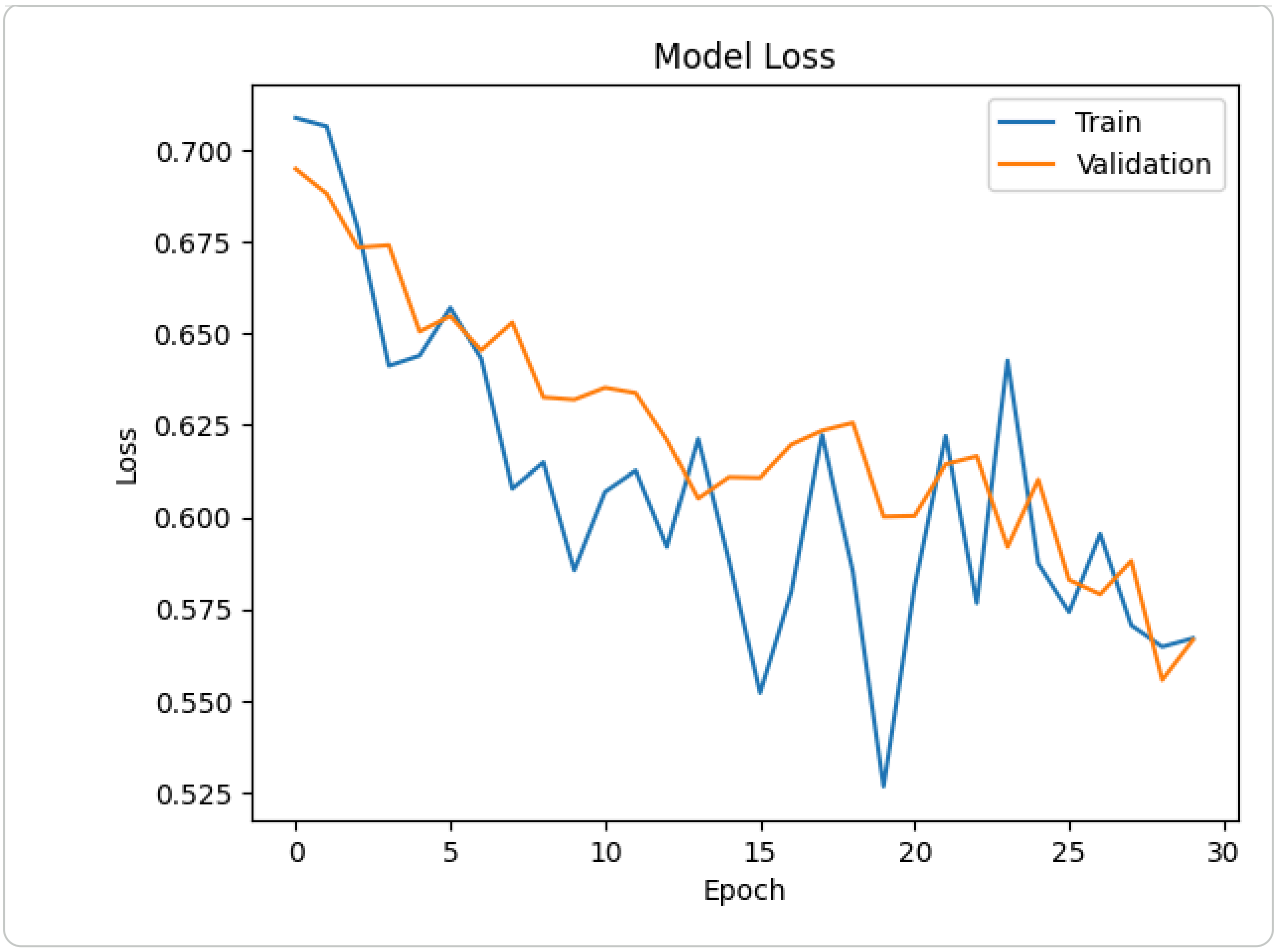
Plot learning curves

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()
```



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

Method 3

Method 3 involved designing a deeper custom convolutional neural network architecture, including regularisation techniques such as dropout, and applying data augmentation during the training to improve generalisation. The model was made to be more complex than the one seen in Method 1 and was designed to ideally extract the more abstract patterns and reduce any overfitting.

Design Explanation

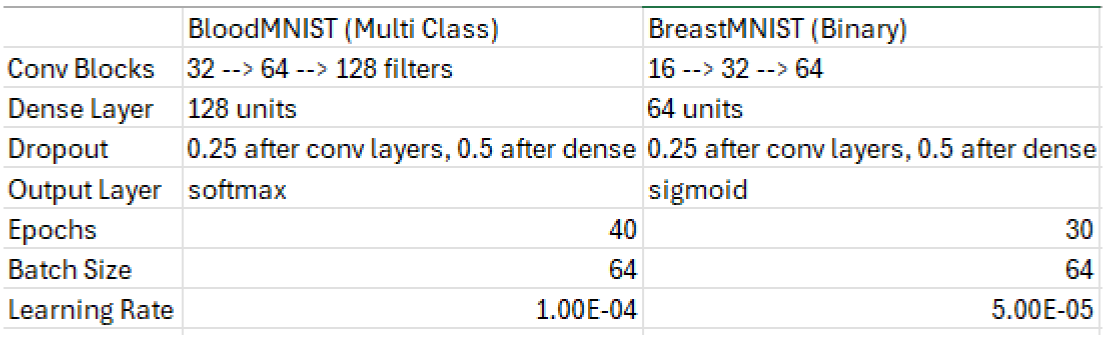
Unlike in Method 1 (a simple CNN), and Method 2 (a shallow dense-only model), Method 3 opted for:

- Three convolutional blocks, with an increasing number of filters (progressing from 16/32 up to 64/128)
- Dropout layers after each block to regularise and prevent overfitting.
- A fully connected layer after flattening, with an additional dropout.

- Data augmentation to allow more variation in the training data
- A different optimiser (RMSprop) was used instead of SGD as in the other two methods. This change was based on the exprimentation of the optimisers and this one seemed to produce more stable and improved results during training.

The model structure however needed to be modified slightly for both datasets, see above in the model definition code for further clarification.

Architectural and Training Differences by the Dataset



	BloodMNIST (Multi Class)	BreastMNIST (Binary)
Conv Blocks	32 --> 64 --> 128 filters	16 --> 32 --> 64
Dense Layer	128 units	64 units
Dropout	0.25 after conv layers, 0.5 after dense	0.25 after conv layers, 0.5 after dense
Output Layer	softmax	sigmoid
Epochs	40	30
Batch Size	64	64
Learning Rate	1.00E-04	5.00E-05

Explanation

- BloodMNIST is more complex (8 classes), which therefore required more model capacity and deeper feature extraction. This is why more filters and a larger dense layer was used.
- BreastMNIST is simpler (binary), and early iterations of similar training parameters to BloodMNIST showed unstable learning. So, the model was lightened and the learning rate was reduced to hopefully improve the convergence stability and reduce the number of fluctuations present in the graphs.

Data Augmentation

As stated in the assignment, data augmentation was to be used in Method 3. The code from this starting example was used for this:

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, axis=0))

This helped improve the generalisation and reduce the overfitting, especially for BloodMNIST.

Performance Results

For clarity of the results. Please see above in this notebook for Method 3 showing the individual epochs variables in the training area and the graphs above showing the