ProjectCore

March 13, 2023

1 Import necessary packages

```
In [1]: #import modules and pytorch libraries
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline
        import seaborn as sns
        import random

        import torch
        import torchvision
        from torch.utils.data import Dataset, DataLoader
        from torch.utils.data import random_split
        from torchvision import datasets
        from torchvision.transforms import ToTensor
        from torchvision.datasets import ImageFolder
        import torchvision.transforms as transforms

        import tensorflow as tf
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, BatchNormaliz
        from tensorflow.keras.layers import Dropout
        print(tf.__version__)
        tf.config.set_visible_devices([], 'GPU')

        # reproducibility
        random.seed(147)
        np.random.seed(147)
        torch.manual_seed(147)
        tf.random.set_seed(147)
```

```
2.11.0
```

```
In [2]: # run this cell if your jupyter notebook kernel has died
        #import os
        #os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'
```

2 Import all images and split for training, validation, and testing

In [3]:
```
#importing training and test dataset
transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=1),  # Convert to grayscale (if needed)
    transforms.Resize((256, 256)),  # Resize the image to (256, 256) pixels
    transforms.ToTensor()  # Convert to a tensor
])

# Load the images from the two folders
# firstly create new folder named 'data' containing yes_output and no_output files
image_set = ImageFolder(root='Br35H/data', transform=transform)

# Define the ratio for each set
train_ratio = 0.8  # 80% for training
val_ratio = 0.1    # 10% for validation
test_ratio = 0.1   # 10% for testing

# Calculate the lengths of each set
train_len = int(len(image_set) * train_ratio)
val_len = int(len(image_set) * val_ratio)
test_len = len(image_set) - train_len - val_len

# Split the dataset using random_split
train_set, val_set, test_set = random_split(image_set, [train_len, val_len, test_len])

batch_size = 32
train_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffle=T
val_loader = torch.utils.data.DataLoader(val_set, batch_size=batch_size)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=batch_size)
```

3 Here I chose to work with tensorflow model for its convenience

3.1 Define a converter here so pytorch dataloader can work with tensorflow

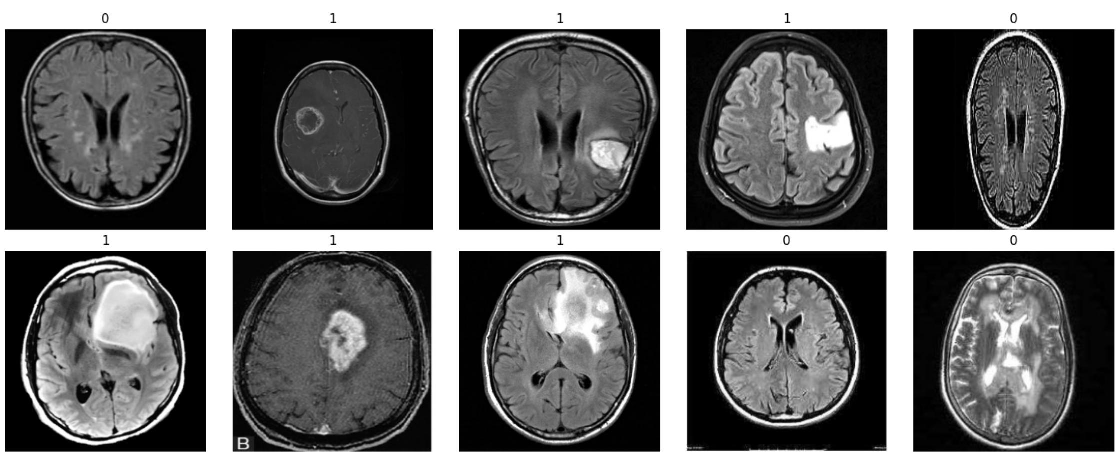
In [4]:
```
# pytorch to tensorflow converter, so that data can be read by tf model
def convert_to_numpy(loader):
    data = []
    labels = []
    for batch in loader:
        images, batch_labels = batch
        data.append(images.numpy())
        labels.append(batch_labels.numpy())
    data = np.concatenate(data, axis=0)
    labels = np.concatenate(labels, axis=0)
    # Reshape the data to (batch_size, height, width, channels)
    data = data.reshape(-1, 256, 256, 1)
    return data, labels
```

```
x_train, y_train = convert_to_numpy(train_loader)
x_val, y_val = convert_to_numpy(val_loader)
x_test, y_test = convert_to_numpy(test_loader)
```

4 Display few images in the folder with corresponding labels

4.1 0: absence of brain tumour, 1: brain tumour positive

In [12]:
```
# Display 10 random original images
n_samples = 10
indices = np.random.randint(x_train.shape[2], size=n_samples)

fig, axes = plt.subplots(2, 5, figsize=(15, 6))
for i, idx in enumerate(indices):
    row = i // 5
    col = i % 5
    axes[row, col].imshow(x_train[idx], cmap='gray')
    axes[row, col].set_title(y_train[idx])
    axes[row, col].axis('off')

plt.tight_layout()
plt.show()
```



5 Define CNN model architecture and compile it

In [7]:
```
#define neural network class (CNN model architecture)
model = Sequential([

# Convolutional layer 1
    Conv2D(32, (3, 3), padding='same', activation = 'relu', input_shape=(256, 256, 1))
```

```python
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Dropout(0.25),

    # Convolutional layer 2
        Conv2D(32, (3, 3), padding='same', activation = 'relu',
               kernel_initializer = 'he_uniform'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),

        Flatten(),

    # Dense layer 1
        Dense(512, activation='relu'),
        Dropout(0.5),

    # Dense layer 3 (output)
        Dense(1, activation='sigmoid')
    ])

    # Compile the model
    model.compile(optimizer = 'sgd', loss='binary_crossentropy', metrics=['accuracy'])
    model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 32)      320

 batch_normalization (BatchN  (None, 256, 256, 32)     128
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 128, 128, 32)     0
 )

 dropout (Dropout)           (None, 128, 128, 32)      0

 conv2d_1 (Conv2D)           (None, 128, 128, 32)      9248

 batch_normalization_1 (Batc  (None, 128, 128, 32)     128
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 64, 64, 32)       0
 2D)

 flatten (Flatten)           (None, 131072)            0
```

```
 dense (Dense)               (None, 512)              67109376

 dropout_1 (Dropout)         (None, 512)              0

 dense_1 (Dense)             (None, 1)                513

=================================================================
Total params: 67,119,713
Trainable params: 67,119,585
Non-trainable params: 128
_________________________________________________________________
```

6 Training the CNN for 14 epochs with validation

In [8]:
```
# training
history = model.fit(x_train, y_train, epochs = 14, batch_size= 32, validation_data=(x_v
```

```
Epoch 1/14


2023-03-08 21:47:58.903530: W tensorflow/tsl/platform/profile_utils/cpu_utils.cc:128] Failed to
2023-03-08 21:47:59.732177: I tensorflow/compiler/xla/service/service.cc:173] XLA service 0x2a4
2023-03-08 21:47:59.732804: I tensorflow/compiler/xla/service/service.cc:181]   StreamExecutor
2023-03-08 21:48:00.143052: I tensorflow/compiler/jit/xla_compilation_cache.cc:477] Compiled cl


75/75 [==============================] - 56s 742ms/step - loss: 1.7454 - accuracy: 0.7429 - val
Epoch 2/14
75/75 [==============================] - 59s 780ms/step - loss: 0.3325 - accuracy: 0.8537 - val
Epoch 3/14
75/75 [==============================] - 56s 741ms/step - loss: 0.2286 - accuracy: 0.9096 - val
Epoch 4/14
75/75 [==============================] - 56s 742ms/step - loss: 0.1589 - accuracy: 0.9388 - val
Epoch 5/14
75/75 [==============================] - 56s 743ms/step - loss: 0.1090 - accuracy: 0.9617 - val
Epoch 6/14
75/75 [==============================] - 51s 681ms/step - loss: 0.0670 - accuracy: 0.9779 - val
Epoch 7/14
75/75 [==============================] - 49s 651ms/step - loss: 0.0559 - accuracy: 0.9808 - val
Epoch 8/14
75/75 [==============================] - 48s 634ms/step - loss: 0.0457 - accuracy: 0.9829 - val
Epoch 9/14
75/75 [==============================] - 49s 647ms/step - loss: 0.0275 - accuracy: 0.9917 - val
Epoch 10/14
75/75 [==============================] - 48s 642ms/step - loss: 0.0207 - accuracy: 0.9937 - val
Epoch 11/14
75/75 [==============================] - 52s 697ms/step - loss: 0.0195 - accuracy: 0.9950 - val
```

```
Epoch 12/14
75/75 [==============================] - 52s 691ms/step - loss: 0.0113 - accuracy: 0.9979 - va
Epoch 13/14
75/75 [==============================] - 77s 1s/step - loss: 0.0098 - accuracy: 0.9983 - val_lo
Epoch 14/14
75/75 [==============================] - 75s 996ms/step - loss: 0.0117 - accuracy: 0.9975 - va
```

7 Evaluate CNN on testing image data

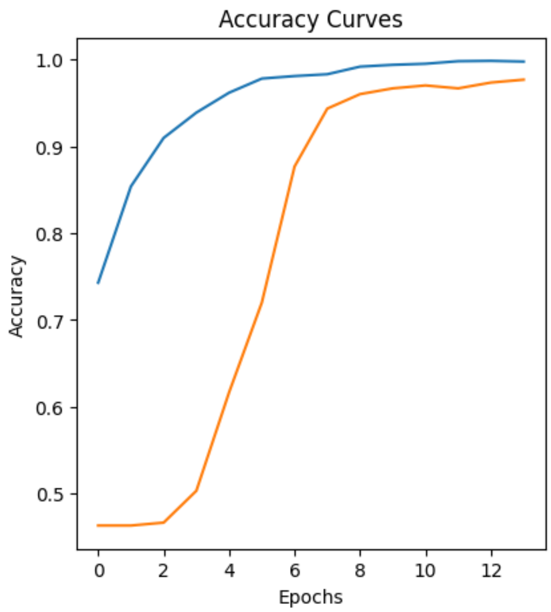
7.1 model displays 98.33% accuracy on unseen data

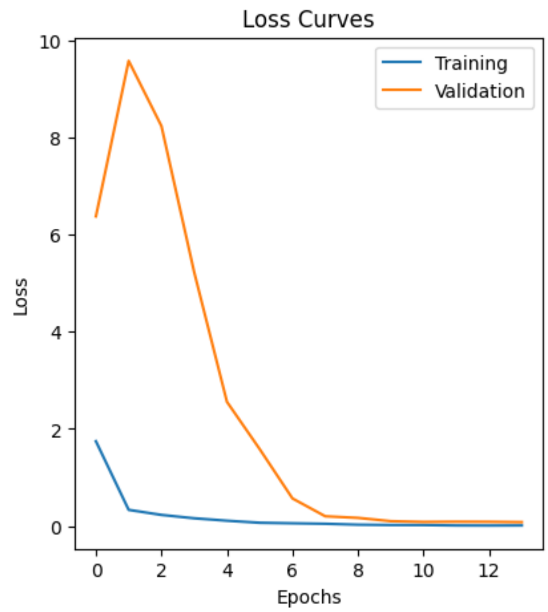
In [9]:
```
# evaluation on testing data
test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
10/10 [==============================] - 1s 138ms/step - loss: 0.0470 - accuracy: 0.9900
```

8 Visualise accuracy and loss values over training epochs

In [10]:
```
# plot accuracy and loss over training epochs
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

# Plot the accuracy and loss curves side by side
plt.figure(figsize=(10,5))
plt.subplot(1, 2, 1) # Create the left subplot
plt.plot(accuracy, label='Training Accuracy')
plt.plot(val_accuracy, label='Validation Accuracy')
plt.title('Accuracy Curves')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')

plt.subplot(1, 2, 2) # Create the right subplot
plt.plot(loss, label='Training')
plt.plot(val_loss, label='Validation')
plt.legend()
plt.title('Loss Curves')
plt.xlabel('Epochs')
plt.ylabel('Loss')

plt.show() # Show the plots
```





9 Save model into HDF5 file or read the file when reopening python

```
In [11]: # Save the entire model to a HDF5 file
         model.save('tumour_detector.h5')
```

```
In [12]: # OR YOU CAN LOAD THE TRAINED MODEL FOR ANALYSIS when reopen python
         model = tf.keras.models.load_model('tumour_detector.h5')
```

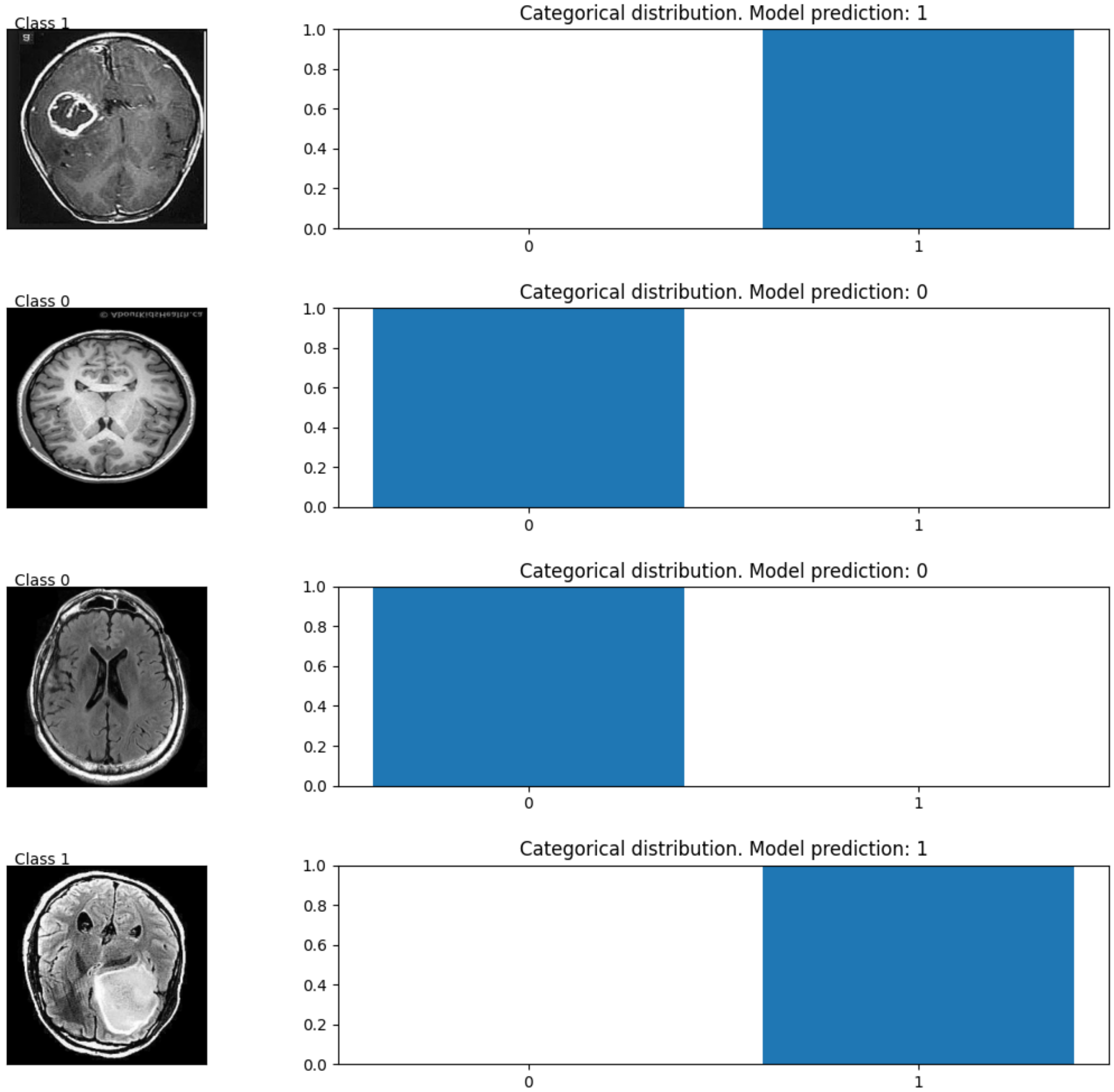
10 Visualise CNN decision making in some of testing images

10.1 Y-axis values represent model confidence on each choices

```
In [13]: # visualise our model's relative confidence (decision making) on testing images
         num_test_images = x_test.shape[0]

         random_inx = np.random.choice(num_test_images, 4)
         random_test_images = x_test[random_inx, ...]
         random_test_labels = y_test[random_inx, ...]

         predictions = model.predict(random_test_images)
         complement = np.ones_like(predictions) - predictions
         predictions = np.concatenate((complement, predictions), axis=1)

         fig, axes = plt.subplots(4, 2, figsize=(16, 12))
         fig.subplots_adjust(hspace=0.4, wspace=-0.2)
```

```
for i, (prediction, image, label) in enumerate(zip(predictions, random_test_images, ra
    axes[i, 0].imshow(np.squeeze(image), cmap = 'gray')
    axes[i, 0].get_xaxis().set_visible(False)
    axes[i, 0].get_yaxis().set_visible(False)
    axes[i, 0].text(10., -1.5, f'Class {label}')
    axes[i, 1].bar(np.arange(len(prediction)), prediction)
    axes[i, 1].set_xticks(np.arange(len(prediction)))
    axes[i, 1].set_title(f"Categorical distribution. Model prediction: {np.argmax(pred

for ax in axes[:,1]:
    ax.set_ylim([0,1])

plt.show()
```

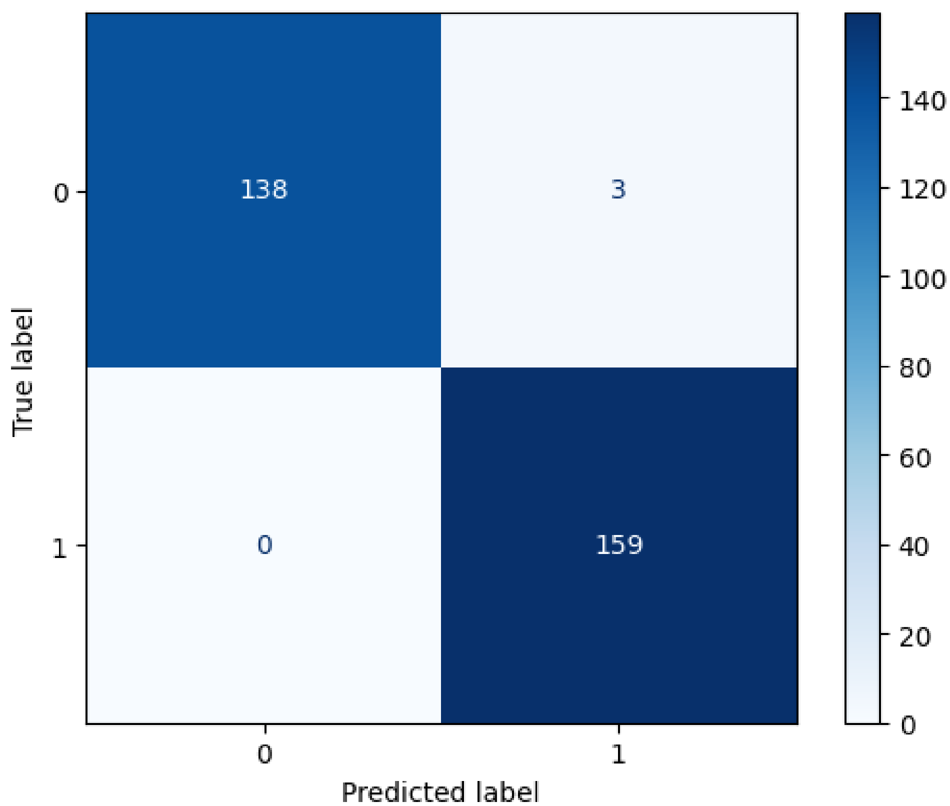
```
1/1 [==============================] - 0s 88ms/step
```



11 Confusion matrix on testing data: 99% accuracy

```
In [14]: # Confusion matrix on testing set to get an overall picture
         from sklearn.metrics import accuracy_score, confusion_matrix
         from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

         predictions = model.predict(x_test)
         predictions = [1 if x>0.5 else 0 for x in predictions]

         accuracy = accuracy_score(y_test, predictions)
         print('Testing Accuracy = %.3f' % accuracy)

         confusion_mtx = confusion_matrix(y_test, predictions)
         disp = ConfusionMatrixDisplay(confusion_matrix=confusion_mtx)
         disp = disp.plot(cmap=plt.cm.Blues)
```

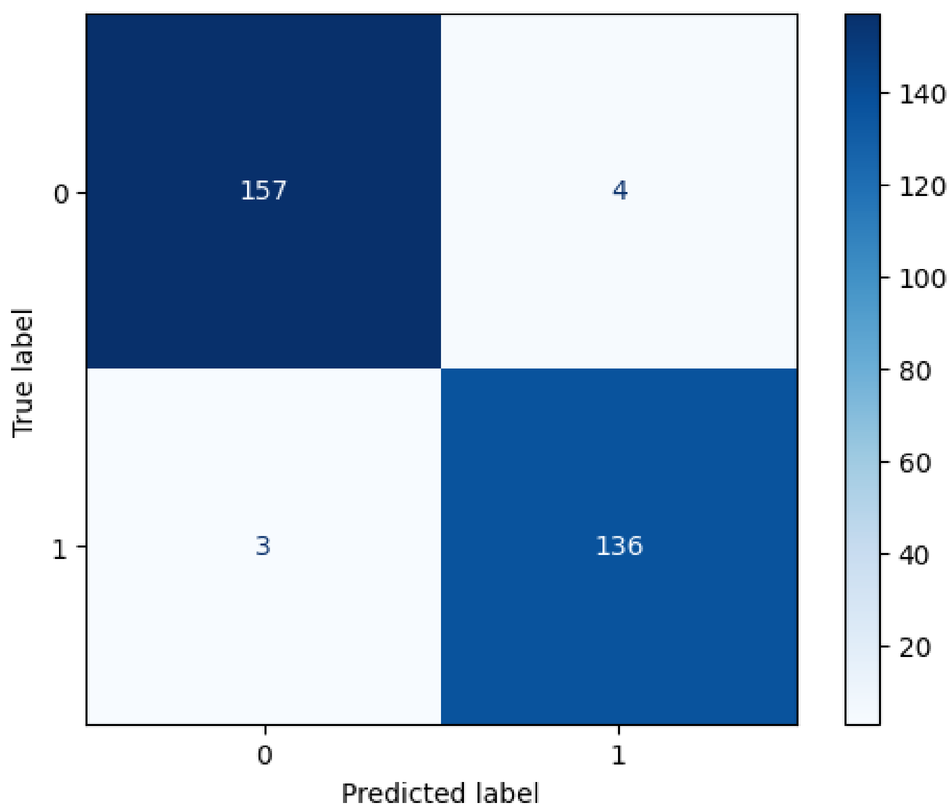
```
10/10 [==============================] - 2s 173ms/step
Testing Accuracy = 0.990
```



12 Confusion matrix for validation data: 97.7% accuracy

```
In [15]: predictions = model.predict(x_val)
         predictions = [1 if x>0.5 else 0 for x in predictions]

         accuracy = accuracy_score(y_val, predictions)
         print('Val Accuracy = %.3f' % accuracy)

         confusion_mtx = confusion_matrix(y_val, predictions)
         disp = ConfusionMatrixDisplay(confusion_matrix=confusion_mtx)
         disp = disp.plot(cmap=plt.cm.Blues)
```

```
10/10 [==============================] - 2s 173ms/step
Val Accuracy = 0.977
```



12.1 Define functions that read an image folder and process images inside just like before. Processed images are ready to plug into model for prediction

```
In [20]: # Define a convert_to_numpy function which reads in image folders and process them
         from PIL import Image
         import os
```

```
def read_images(pred_path, transform):
    img_filenames = os.listdir(pred_path)
    img_width, img_height = 256, 256

    # Create a numpy array to store the images:
    imgs = np.zeros((len(img_filenames), img_width, img_height), dtype=np.float32)

    # Loop through the image filenames, load each image, and preprocess it:
    for i, filename in enumerate(img_filenames):
        img = Image.open(os.path.join(pred_path, filename))
        img = transform(img)
        imgs[i] = img

    # Reshape the data to (batch_size, height, width, channels)
    imgs = imgs.reshape(-1, img_width, img_height, 1)
    return imgs

# Get prediction images from pred_output image folder
pred_path = 'Br35H/pred_output'
img_filenames = os.listdir(pred_path)
img_width, img_height = 256, 256
pred_imgs = read_images(pred_path, transform)
```

13 Visualise model predictions of some images from folder 'pred_output'

In [21]:
```
# visualise CNN decision making by displaying confidences
num_test_images = pred_imgs.shape[0]
random_inx = np.random.choice(num_test_images, 4)
random_test_images = pred_imgs[random_inx, ...]

predictions = model.predict(random_test_images)
complement = np.ones_like(predictions) - predictions
predictions = np.concatenate((complement, predictions), axis=1)

fig, axes = plt.subplots(4, 2, figsize=(16, 12))
fig.subplots_adjust(hspace=0.4, wspace=-0.2)

for i, (prediction, image) in enumerate(zip(predictions, random_test_images)):
    axes[i, 0].imshow(np.squeeze(image), cmap = 'gray')
    axes[i, 0].get_xaxis().set_visible(False)
    axes[i, 0].get_yaxis().set_visible(False)
    axes[i, 1].bar(np.arange(len(prediction)), prediction)
    axes[i, 1].set_xticks(np.arange(len(prediction)))
    axes[i, 1].set_title(f"Categorical distribution. Model prediction: {np.argmax(pre
```

```
for ax in axes[:,1]:
    ax.set_ylim([0,1])

plt.show()
```

```
1/1 [==============================] - 0s 117ms/step
```