CNN Model 1

In [ ]:
```
## Reproducibility
## Random seed given
import random ## import the random library
random.seed(10) ## Setting the seed to get the same answer no matter how many
 times and who runs the model
```

In [ ]:
```
## Downloading specific libraries
import numpy as np ## Library that enables linear functions
import pandas as pd ##  # Enables data processing
import glob ## returns an array of filenames that match a pattern
import cv2 ## helps add labels to image classifications
import matplotlib.pyplot as plt ## library for producing figures
from keras.preprocessing.image import ImageDataGenerator, load_img, img_to_arr
ay, array_to_img ## importing image processing packages from
## keras
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout ## Impo
rt libraries for model building
```

In [ ]:
```
## Read the train csv file
train_dir='/kaggle/input/siim-isic-melanoma-classification/jpeg/train/' ## ass
igning a name to the location of the train images
train=pd.read_csv('/kaggle/input/siim-isic-melanoma-classification/train.csv')
## assigning a name to the location of the CSV file

## Read the test csv file
test_dir='/kaggle/input/siim-isic-melanoma-classification/jpeg/test/' ## assig
ning a name to the location of the test images
test=pd.read_csv('/kaggle/input/siim-isic-melanoma-classification/test.csv') #
# assigning a name to the location of the CSV file
```

In [ ]:
```
## Finding the unique patient ids from train csv file
print(f"The total patient ids are {train['patient_id'].count()}, from those th
e unique ids are {train['patient_id'].value_counts().shape[0]} ")

## Finding the unique patient ids from test csv file
print(f"The total patient ids are {test['patient_id'].count()}, from those the
unique ids are {test['patient_id'].value_counts().shape[0]} ")
```

In [ ]:
```
train['path'] = train_dir + train.image_name + ".jpg" ## adding the location o
f the image to the row for the train data set
train.head() ## showing the first 5 lines of the train data set, note the "pat
h" coloumn

test['path'] = test_dir + test.image_name + ".jpg"  ## adding the location of
 the image to the row for the test data set
test.head()  ## showing the first 5 lines of the test data set, note the "pat
h" coloumn
```

some images

In [ ]:
```
## Class Distribution
train.target.value_counts() ## Count the number of images that were classified as malinnent or non malignent
```

In [ ]:
```
df_0=train[train['target']==0].sample(1000) ## produce a data frame using 1000 images from the train data set where the target equals zero
df_1=train[train['target']==1] ## produce a data frame using all the images from the test data set where the target equals 584
train=pd.concat([df_0,df_1]) ## create a new dataset using the smaller training data set
train=train.reset_index() ## making sure the new "train" data set is being used for the model
```

In [ ]:
```
train.shape ## how many observations and variables are in the training set being used for the model
```

In [ ]:
```
train.head() ## First 5 rows of the new train set
```

In [ ]:
```
# we will resize the given images to 150 x 150 size images for faster processing
IMG_DIM = (150, 150) ## changing the image dimensions
```

In [ ]:
```
from sklearn.model_selection import train_test_split ## importing the train test split function
X_train, X_val, y_train, y_val = train_test_split(train, train.target, test_size=0.2, random_state=42) ## taking 20% of the training data set
```

In [ ]:
```
train_files = X_train.path ## Image path for the training data set
val_files = X_val.path ## Image path for the validation data set

train_imgs = [img_to_array(load_img(img, target_size=IMG_DIM)) for img in train_files] ## load images using load_img function from keras
## preprocessing using the target_size function
validation_imgs = [img_to_array(load_img(img, target_size=IMG_DIM)) for img in val_files] ## using the img_to_array will tranform the loaded image to an array

train_imgs = np.array(train_imgs) ## converting the list of arrays to array for the training dataset
train_labels = y_train

validation_imgs = np.array(validation_imgs) ## converting the list of arrays to array for the validation dataset
val_labels = y_val


print('Train dataset shape:', train_imgs.shape,
      '\tValidation dataset shape:', validation_imgs.shape)
```

In [ ]:
```
## Scale Images
## scale each image with pixel values between (0, 255) to values between (0,
 1) because deep learning models work really
## well with small input values.
train_imgs_scaled = train_imgs.astype('float32')

validation_imgs_scaled  = validation_imgs.astype('float32')

# divide the pixels by 255 to scale the pixels between 0 and 1
train_imgs_scaled /= 255
validation_imgs_scaled /= 255

print(train_imgs[0].shape)

array_to_img(train_imgs[0]) ## using the array_to_img function will convert th
e given array to image
```

In [ ]:
```
# setup basic configuration
batch_size = 30 ## indicating the total number of images passed to the model p
er iteration
num_classes = 2
epochs = 30 ## establishing the training time
input_shape = (150, 150, 3)
```

In [ ]:
```
import random ## import the random library
random.seed(10) ## Setting the seed to get the same answer no matter how many
 times and who runs the model

from keras.models import Sequential ## importing the sequential library
from keras import optimizers ## importing optimizers

model = Sequential() ## creating and instance of Sequential

model.add(Conv2D(16, kernel_size=(3, 3), activation='relu',
                 input_shape=input_shape))
# Pooling layer used here will select the largest values on the feature maps a
nd use these as inputs to subsequent layers
model.add(MaxPooling2D(pool_size=(2, 2)))


# another set of Convolutional & Max Pooling layers
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
# Finally the Dense Layer
model.add(Dense(512, activation='relu'))
# sigmoid function here will help perform binary classification
model.add(Dense(1, activation='sigmoid'))


model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(),
              metrics=['accuracy'])

model.summary()
```

In [ ]:
```
history = model.fit(x=train_imgs_scaled, y=train_labels,
                    validation_data=(validation_imgs_scaled, val_labels),
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1)
```

In [ ]:
```
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
t = f.suptitle('CNN Model 1', fontsize=12)
f.subplots_adjust(top=0.85, wspace=0.3)

epoch_list = list(range(1,31))
ax1.plot(epoch_list, history.history['accuracy'], label='Train Accuracy')
ax1.plot(epoch_list, history.history['val_accuracy'], label='Validation Accura
cy')
ax1.set_xticks(np.arange(0, 31, 5))
ax1.set_ylabel('Accuracy Value')
ax1.set_xlabel('Epoch')
ax1.set_title('Accuracy')
l1 = ax1.legend(loc="best")

ax2.plot(epoch_list, history.history['loss'], label='Train Loss')
ax2.plot(epoch_list, history.history['val_loss'], label='Validation Loss')
ax2.set_xticks(np.arange(0, 31, 5))
ax2.set_ylabel('Loss Value')
ax2.set_xlabel('Epoch')
ax2.set_title('Loss')
l2 = ax2.legend(loc="best")
```

The model ran shows that there is a level of overfitting, the train accuracy continues to increase until it gets to 100%, the validation accuracy begins to fall of at 70-75%. The train loss declined at around 75% right down to zero, while the validation loss continues to increase. Overfitting is a result of not enough images to train the model.