Skin-Cancer MNIST Training Using CNN

Dataset:

I downloaded the dataset from the link provided in the Kaggle . I downloads two zip files of directory of images which specifies skin cancer and extracted them into single folder which is HAM10000_images_part_1 the total images comprise of 10015 images and I also downloaded a metadata csv file which indicates info about all the 10015 images in the above directory

Number of unique classes:

There are a total of 7 unique classes mentioned in csv file they are

bkl: Benign keratosis

nv: Melanocytic nevi

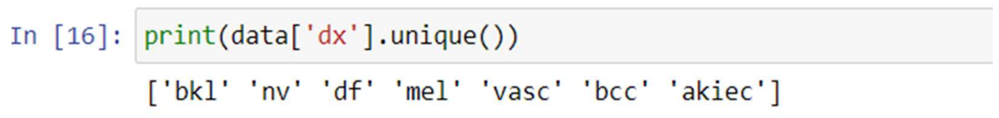
df: Dermatofibroma

mel: Melanoma

vasc: Vascular skin lesions

bcc: Basal cell carcinoma

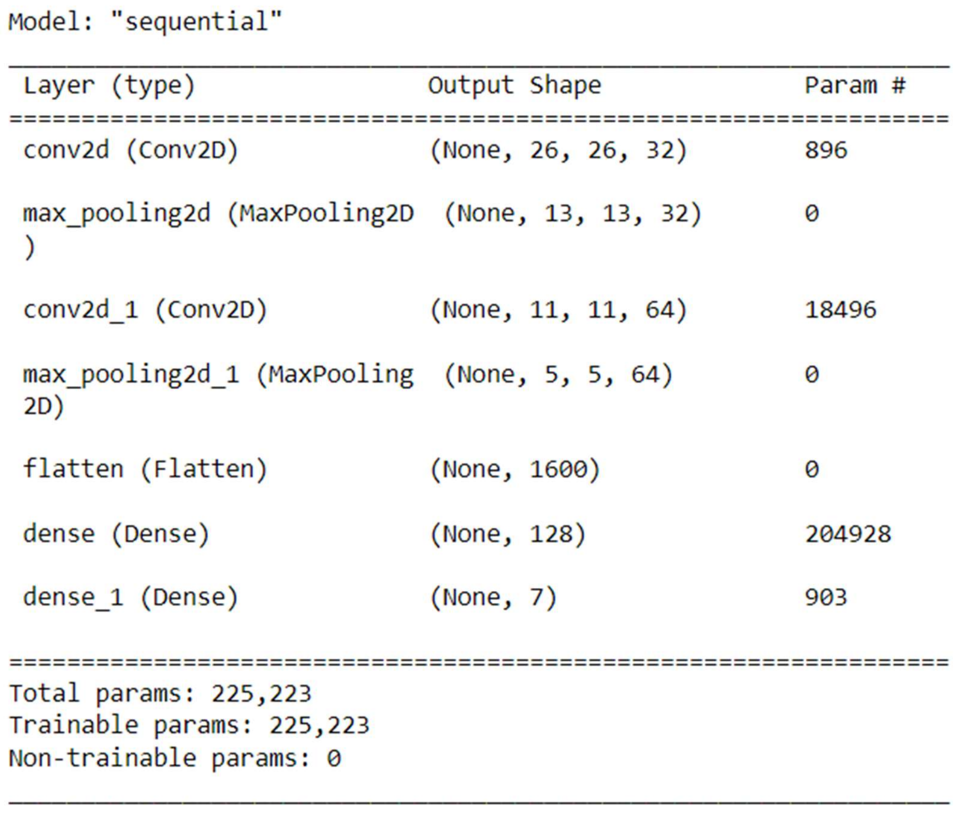
akiec: Actinic Keratoses and Intraepithelial Carcinoma



```
In [16]: print(data['dx'].unique())
         ['bkl' 'nv' 'df' 'mel' 'vasc' 'bcc' 'akiec']
```

Description of Model:

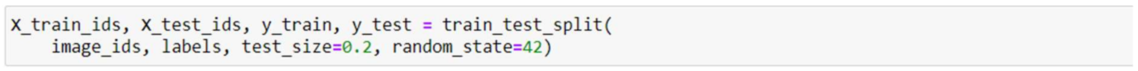
I used CNN model to train the data with each input as an image of target size (20,20) the model consists of two convolutional layers and two max pooling layer alternatively and followed by one flatten and two dense layer . to classify the skin cancer we used the softmax fu nction for classification and used ReLu activation function for convolution layer. Each filter size is (3,3) and max pool filter size is (2,2) the total model is



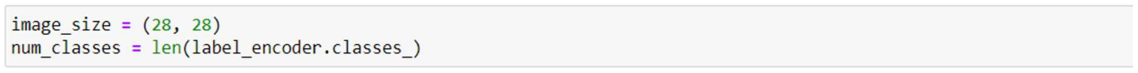
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        896

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 128)               204928

 dense_1 (Dense)             (None, 7)                 903

=================================================================
Total params: 225,223
Trainable params: 225,223
Non-trainable params: 0
_________________________________________________________________
```

Training the model:

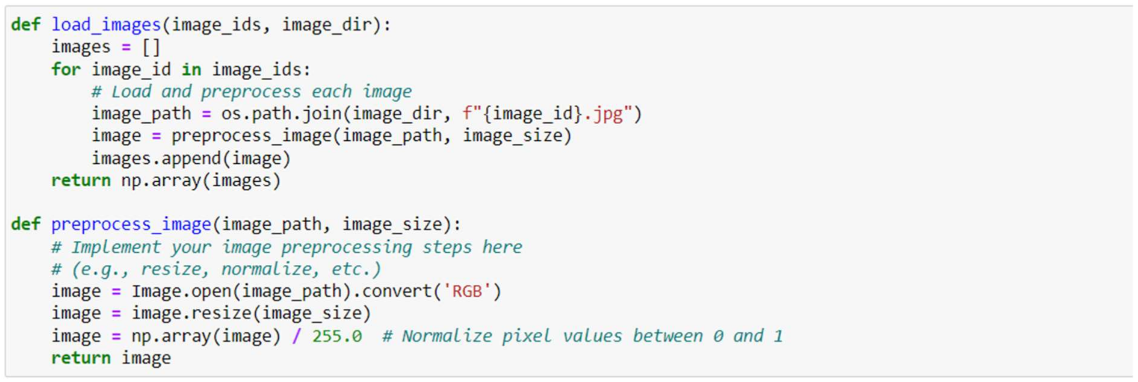
First we load the images from the directory and preprocess the image it is done by defining two functions and path to the directory



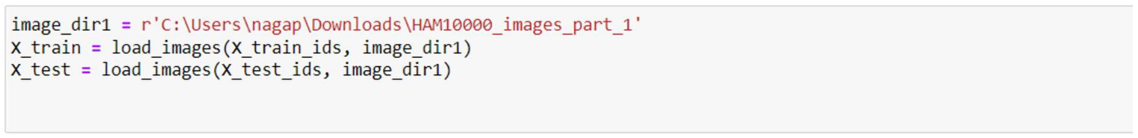
```
X_train_ids, X_test_ids, y_train, y_test = train_test_split(
    image_ids, labels, test_size=0.2, random_state=42)
```



```
image_size = (28, 28)
num_classes = len(label_encoder.classes_)
```



```
def load_images(image_ids, image_dir):
    images = []
    for image_id in image_ids:
        # Load and preprocess each image
        image_path = os.path.join(image_dir, f"{image_id}.jpg")
        image = preprocess_image(image_path, image_size)
        images.append(image)
    return np.array(images)

def preprocess_image(image_path, image_size):
    # Implement your image preprocessing steps here
    # (e.g., resize, normalize, etc.)
    image = Image.open(image_path).convert('RGB')
    image = image.resize(image_size)
    image = np.array(image) / 255.0  # Normalize pixel values between 0 and 1
    return image
```



```
image_dir1 = r'C:\Users\nagap\Downloads\HAM10000_images_part_1'
X_train = load_images(X_train_ids, image_dir1)
X_test = load_images(X_test_ids, image_dir1)
```

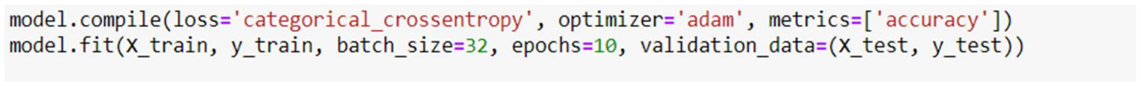
I gave parameters to train the model they are loss='categorical_crossentropy'

optimizer='adam'

metrics='accuracy'

epochs = 10

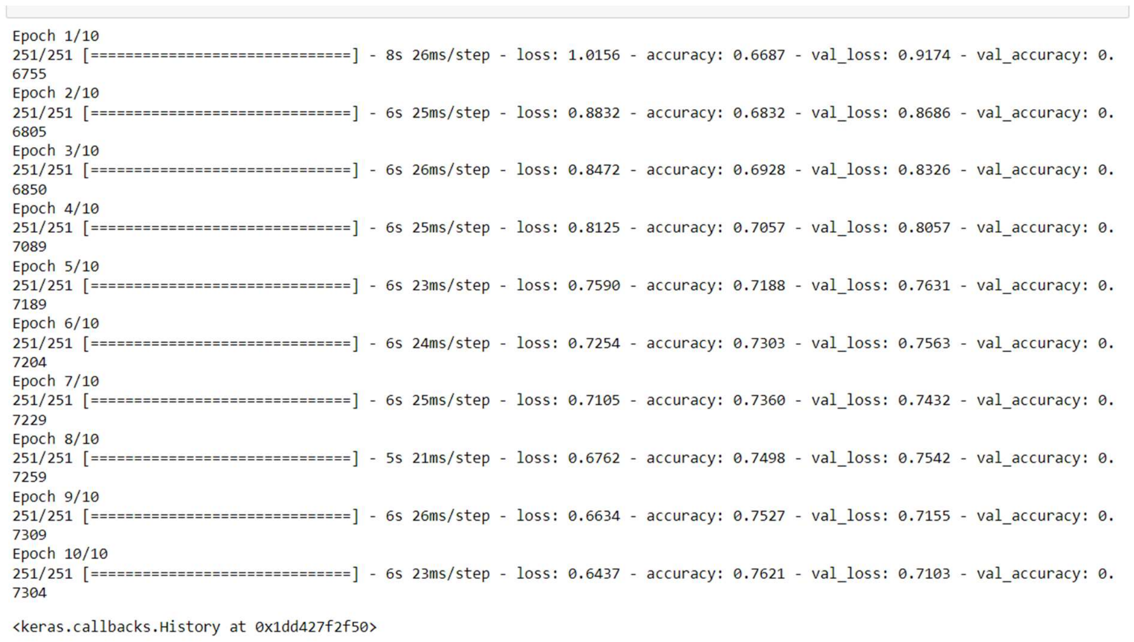
batch size= 32



```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=32, epochs=10, validation_data=(X_test, y_test))
```

Results:

The training accuracy is 0.7621 and loss is 0.6437 and validation accuracy is 0.7304 and validation loss is 0.7103



```
Epoch 1/10
251/251 [==============================] - 8s 26ms/step - loss: 1.0156 - accuracy: 0.6687 - val_loss: 0.9174 - val_accuracy: 0.
6755
Epoch 2/10
251/251 [==============================] - 6s 25ms/step - loss: 0.8832 - accuracy: 0.6832 - val_loss: 0.8686 - val_accuracy: 0.
6805
Epoch 3/10
251/251 [==============================] - 6s 26ms/step - loss: 0.8472 - accuracy: 0.6928 - val_loss: 0.8326 - val_accuracy: 0.
6850
Epoch 4/10
251/251 [==============================] - 6s 25ms/step - loss: 0.8125 - accuracy: 0.7057 - val_loss: 0.8057 - val_accuracy: 0.
7089
Epoch 5/10
251/251 [==============================] - 6s 23ms/step - loss: 0.7590 - accuracy: 0.7188 - val_loss: 0.7631 - val_accuracy: 0.
7189
Epoch 6/10
251/251 [==============================] - 6s 24ms/step - loss: 0.7254 - accuracy: 0.7303 - val_loss: 0.7563 - val_accuracy: 0.
7204
Epoch 7/10
251/251 [==============================] - 6s 25ms/step - loss: 0.7105 - accuracy: 0.7360 - val_loss: 0.7432 - val_accuracy: 0.
7229
Epoch 8/10
251/251 [==============================] - 5s 21ms/step - loss: 0.6762 - accuracy: 0.7498 - val_loss: 0.7542 - val_accuracy: 0.
7259
Epoch 9/10
251/251 [==============================] - 6s 26ms/step - loss: 0.6634 - accuracy: 0.7527 - val_loss: 0.7155 - val_accuracy: 0.
7309
Epoch 10/10
251/251 [==============================] - 6s 23ms/step - loss: 0.6437 - accuracy: 0.7621 - val_loss: 0.7103 - val_accuracy: 0.
7304

<keras.callbacks.History at 0x1dd427f2f50>
```

Reason to use CNN:

The cnn is best to train images and videoes it is eacy to preprocess and train it . it also yield best expected results