Realistic Image Classification

Păpușoi Rareș - 232

Models Used

1. SVM (Support Vector Machine) - 0.457 public score, 0.420 private score

2. CNN (Convolutional Neural Network) - 0.760 public score, 0.751 private score

Loading the data

1. SVM

The data was loaded using Pandas and numpy. The pixel values of each image were normalized from a range of [0, 255] to [0, 1]. I also resized the image to 10x10 pixels, as the original 80x80 would have been to much for the model. The images were also flattened. Function to load train and validation data:

```
def load_and_resize_data(directory, labels_df, target_size=(10, 10)):
    images = []
    labels = []
    for _, row in labels_df.iterrows():
        image_path = os.path.join(directory, f"{row['image_id']}.png")
        img = load_img(image_path, target_size=target_size)
        img_array = img_to_array(img) / 255.0
        images.append(img_array.flatten())
        labels.append(row['label'])
    return np.array(images), np.array(labels)
```

2. CNN

The data was loaded using Pandas and numpy. The pixel values of each image were normalized from a range of [0, 255] to [0, 1]. Function to load train and validation data:

```
def load_image_and_label(image_path, label, target_size):
    img = load_img(image_path, target_size=target_size)
    img_array = img_to_array(img) / 255.0
    return img_array, label

def load_dataset(directory, labels_df, target_size):
    images = []
    labels = []
    for index, row in labels_df.iterrows():
        image_path = os.path.join(directory, f"{row['image_id']}.png")
        image, label = load_image_and_label(image_path, row['label'], target_size)
        images.append(image)
```

```
            labels.append(label)
        return np.array(images), np.array(labels)
```

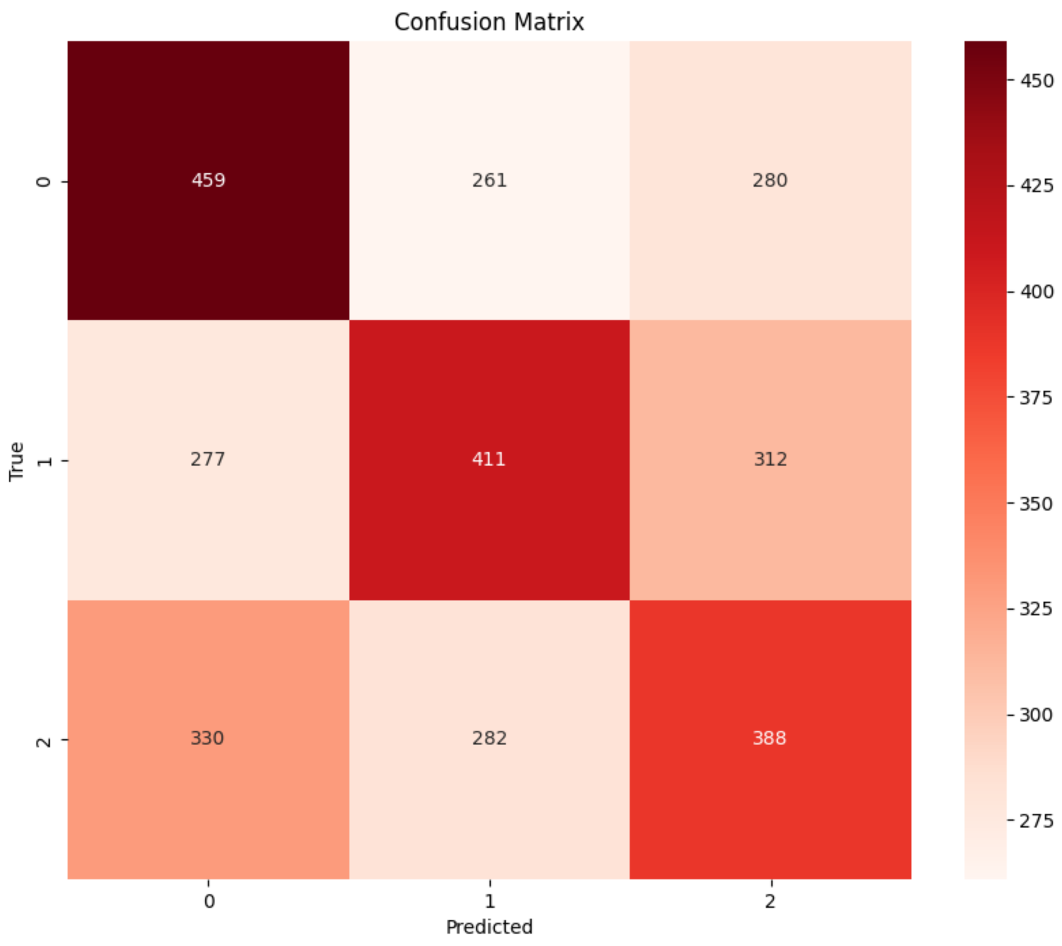
Model Training

1. SVM

```
svm_classifier = SVC(kernel='linear', decision_function_shape='ovr')
svm_classifier.fit(train_images, train_labels)
predictions = svm_classifier.predict(validation_images)
validation_accuracy = accuracy_score(validation_labels, predictions)
print(validation_accuracy)
```

Output:

```
0.41933333333333334
```



For the submission, I trained the model on the train and validation data combined.

```
train_validation_images = np.concatenate((train_images, validation_images))
train_validation_labels = np.concatenate((train_labels, validation_labels))
```

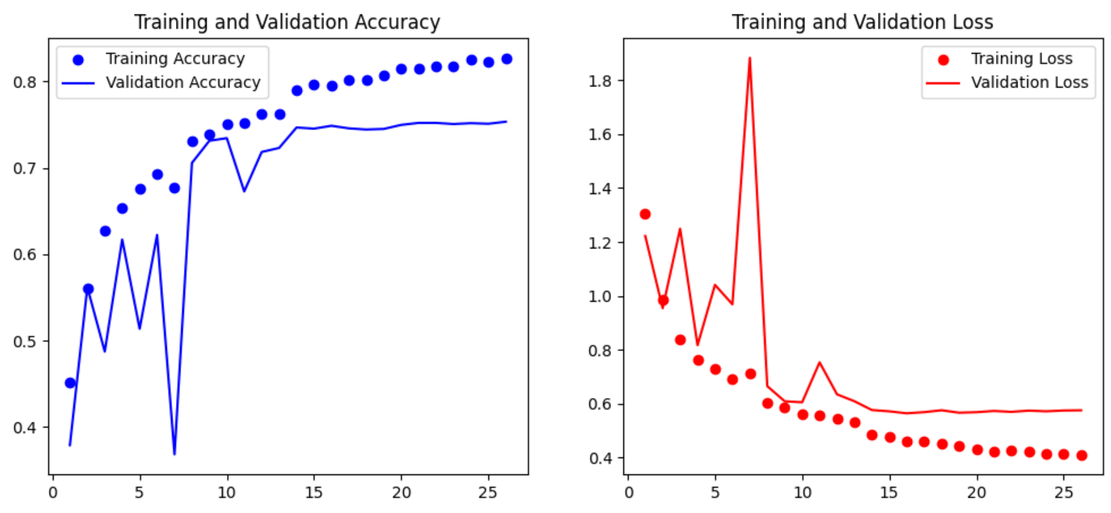
2. CNN

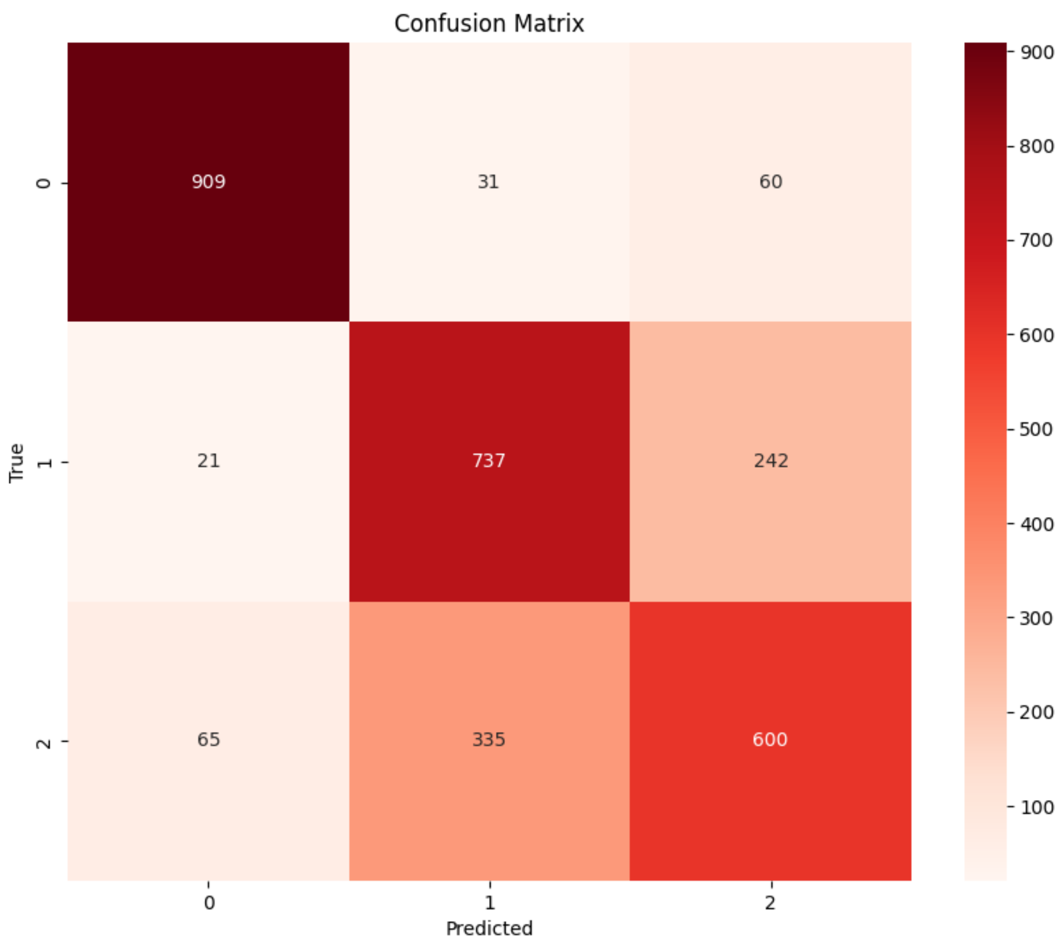
```
def create_model(input_shape=(80, 80, 3), num_classes=3):
    model = Sequential([
        Input(shape=input_shape),

        Conv2D(64, (3, 3), padding='same', activation='relu'),
        BatchNormalization(),
        Conv2D(64, (3, 3), padding='same', activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(128, (3, 3), padding='same', activation='relu'),
        BatchNormalization(),
        Conv2D(128, (3, 3), padding='same', activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(256, (3, 3), padding='same', activation='relu'),
        BatchNormalization(),
        Conv2D(256, (3, 3), padding='same', activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Flatten(),

        Dense(256, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),

        Dense(num_classes, activation='softmax')
    ])
    return model

model = create_model()

learning_rate = 0.001
optimizer = Adam(learning_rate=learning_rate)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3,
min_lr=0.00001)
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=100, validation_data=
(validation_images, validation_labels), callbacks=[reduce_lr, early_stopping])
```

Last epoch output:

```
Epoch 29/100
329/329 ━━━━━━━━━━━━━━━━━━━━ 20s 48ms/step - accuracy: 0.8220 - loss: 0.4067 -
val_accuracy: 0.7537 - val_loss: 0.5696 - learning_rate: 1.0000e-05
```

Because of the early stopping callback, the model stopped training at epoch 29, as the validation loss did not improve for 10 epochs.





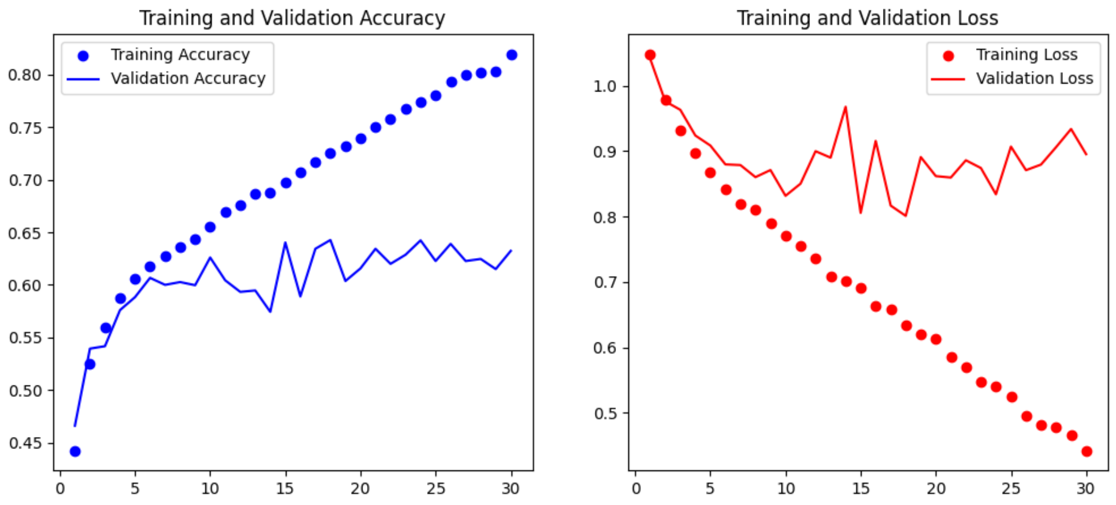
	0	1	2
0	909	31	60
1	21	737	242
2	65	335	600

CNN Evolution

The first model I tried was:

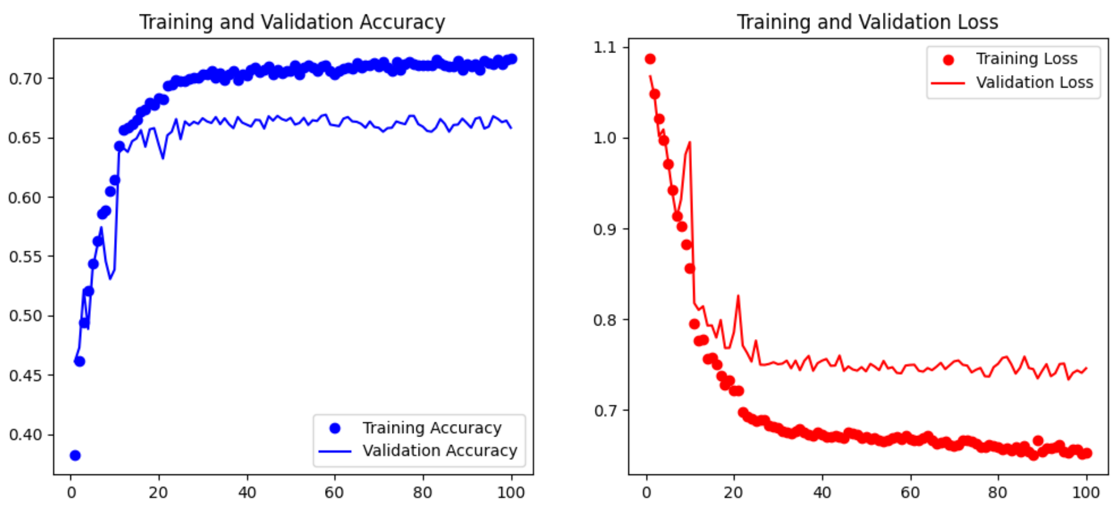
```
def create_model(input_shape=(80, 80, 3), num_classes=3):
    model = Sequential([
        Input(shape=(80, 80, 3)),
        Conv2D(32, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(4, 4)),
        Dropout(0.25),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(4, 4)),
        Dropout(0.25),
        Flatten(),
        Dense(128, activation='relu'),
        Dense(3, activation='softmax')
    ])
    return model

model = create_model()

model.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy', metrics=
['accuracy'])

history = model.fit(train_images, train_labels, epochs=30, validation_data=
(validation_images, validation_labels))
```

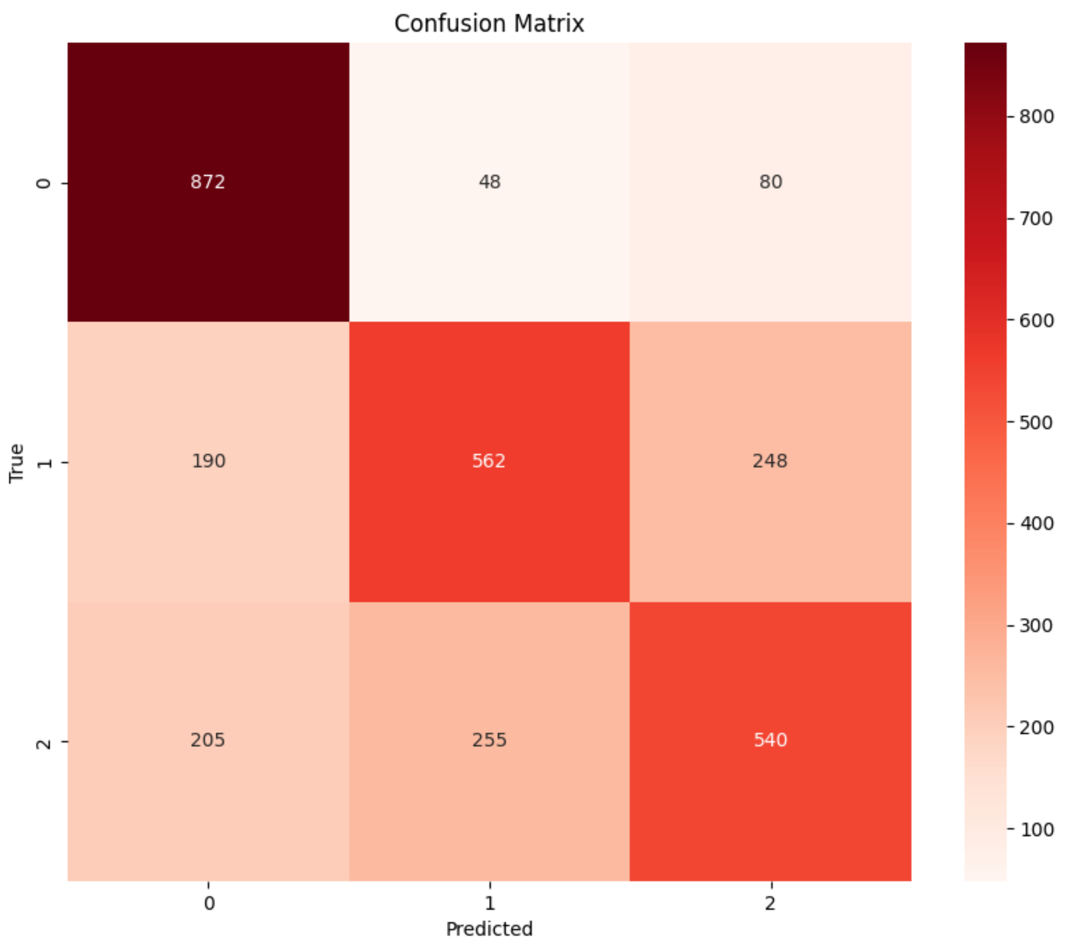
The model had a validation accuracy of around 0.62.



Then followed the model I first submitted. It had a validation accuracy of around 0.66 and a score of 0.672 on the public test set. I introduced dropout and the learning rate reduction callback. I also modified the architecture of the model.

```
def create_model(input_shape=(80, 80, 3), num_classes=3):
    model = Sequential([
        Input(shape=(80, 80, 3)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(4, 4)),
        Dropout(0.25),
        Flatten(),

        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(3, activation='softmax')
    ])
    return model

model = create_model()

learning_rate = 0.001
optimizer = Adam(learning_rate=learning_rate)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3,
min_lr=0.00001)

model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=100, validation_data=
(validation_images, validation_labels), callbacks=[reduce_lr])
```





Afterwards, I started to experiment with data augmentation. I used the ImageDataGenerator from Keras to generate new images from the existing train set.

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

Unfortunately, I saw no improvement in the model's performance. The model I used for my first submission only had a 0.55 validation accuracy. I trained a few different models with data augmentation, but none of them performed better than the model I first submitted, so I thought that in order to improve performance, I should return to trying different architectures, with different hyperparameters.

I first tried a different activation function, gelu, and introduced the early stopping callback.

```
def create_model(input_shape=(80, 80, 3), num_classes=3):
    model = Sequential([
        Input(shape=input_shape),
        Conv2D(64, (3, 3), activation='gelu'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(64, (3, 3), activation='gelu'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(64, (3, 3), activation='gelu'),
        MaxPooling2D(pool_size=(4, 4)),
        Dropout(0.25),
        Flatten(),

        Dense(64, activation='gelu'),
        Dropout(0.5),
        Dense(3, activation='softmax')
    ])
    return model

model = create_model()

learning_rate = 0.001
optimizer = Adam(learning_rate=learning_rate)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3,
min_lr=0.00001)
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=100, validation_data=
(validation_images, validation_labels), callbacks=[reduce_lr, early_stopping])
```

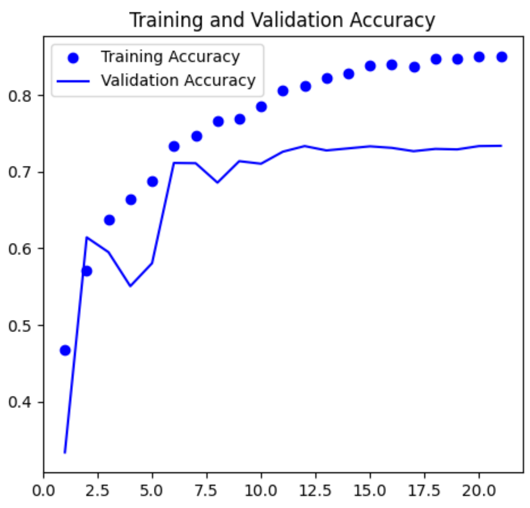
The model had a validation accuracy of around 0.66, so no major improvement was seen. I thought that in order to prevent overfitting, I should use a regularization technique, so I introduced L2 regularization.

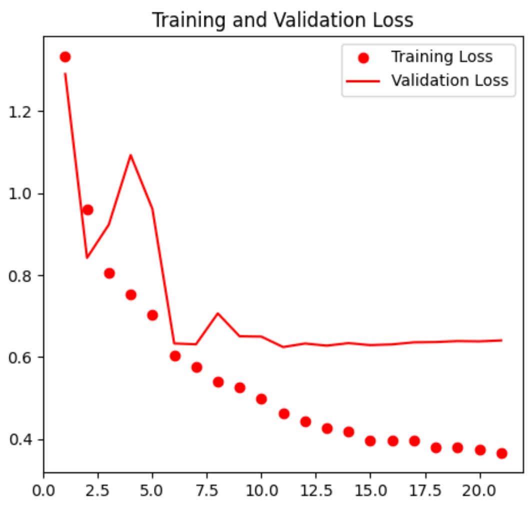
```
def create_model(input_shape=(80, 80, 3), num_classes=3):
    model = Sequential([
        Input(shape=input_shape),
        Conv2D(64, (3, 3), activation='gelu', kernel_regularizer=l2(0.001)),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(64, (3, 3), activation='gelu', kernel_regularizer=l2(0.001)),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(64, (3, 3), activation='gelu', kernel_regularizer=l2(0.001)),
        MaxPooling2D(pool_size=(4, 4)),
        Dropout(0.25),
        Flatten(),

        Dense(128, activation='gelu', kernel_regularizer=l2(0.001)),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])
    return model
```

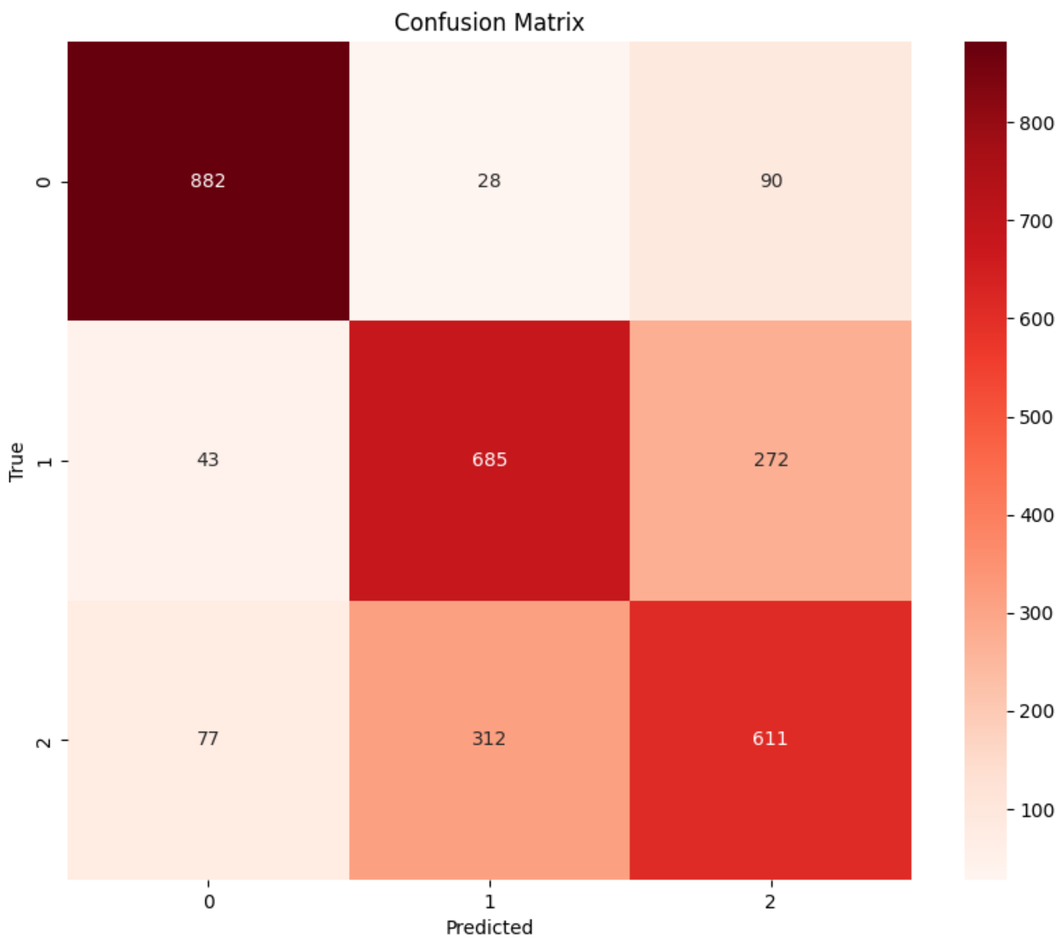
This time, the validation accuracy hovered around 0.65, and I thought that the model was too simple, so I adjusted the architecture. I also added batch normalization.

```
def create_model(input_shape=(80, 80, 3), num_classes=3):
    model = Sequential([
        Input(shape=input_shape),

        Conv2D(64, (3, 3), activation='relu'),
        BatchNormalization(),
        Conv2D(64, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(128, (3, 3), activation='relu'),
        BatchNormalization(),
        Conv2D(128, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),
        Flatten(),

        Dense(256, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])
    return model

model = create_model()

learning_rate = 0.001
optimizer = Adam(learning_rate=learning_rate)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3,
min_lr=0.00001)
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=100, validation_data=
(validation_images, validation_labels), callbacks=[reduce_lr, early_stopping])
```

This was the first time since the first submission that I have seen improvement, the validation accuracy reached 0.72.







I discovered the padding parameter in the convolutional layers. Adding it did not improve or affect the model's performance. I tried different dropout rates for the convolutional layers:

Dropout rate	Validation accuracy	Validation loss
0.25	0.722	0.66
0.5	0.720	0.67
0.75	0.695	0.70

In order to improve the model's performance, I added more convolutional layers, again, since last time this was the change that improved the model's performance. This is how I reached the final model. Different dropout rates were tested for the convolutional layers:

Dropout rate	Validation accuracy	Validation loss
0.25	0.753	0.57
0.5	0.752	0.58
0.75	0.728	0.60

It's clear that the 0.75 dropout rate is the worst (in both cases). I stuck with the 0.25 dropout rate. Throughout the history of the training, these values above were the ones I saw most often. This is the final model I used for the final submission:

```
def create_model(input_shape=(80, 80, 3), num_classes=3):
    model = Sequential([
        Input(shape=input_shape),

        Conv2D(64, (3, 3), padding='same', activation='relu'),
        BatchNormalization(),
        Conv2D(64, (3, 3), padding='same', activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(128, (3, 3), padding='same', activation='relu'),
        BatchNormalization(),
        Conv2D(128, (3, 3), padding='same', activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(256, (3, 3), padding='same', activation='relu'),
        BatchNormalization(),
        Conv2D(256, (3, 3), padding='same', activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),
        Flatten(),

        Dense(256, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),

        Dense(num_classes, activation='softmax')
    ])
    return model

model = create_model()

learning_rate = 0.001
optimizer = Adam(learning_rate=learning_rate)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3,
min_lr=0.00001)
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=100, validation_data=
(validation_images, validation_labels), callbacks=[reduce_lr, early_stopping])
```

The details about the final model can be found here.