Classify_Facial_Expressions

July 28, 2024

1 Read folder from Drive

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

2 Unzip data file

[ ]:
```
from zipfile import ZipFile
file_name = 'drive/MyDrive/emotion/data.zip'
with ZipFile(file_name,'r') as zip:
  zip.extractall()
  print('Done')
```

```
Done
```

3 Importing Libraries

[ ]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os
import tensorflow as tf
%matplotlib inline

from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing.image import load_img, img_to_array
from keras.layers import Input, Conv2D, Flatten, GlobalAveragePooling2D, Dense,␣
↪Dropout, BatchNormalization, Activation, MaxPooling2D , AveragePooling2D
from keras.models import Model, Sequential
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping
from keras.utils import plot_model
from keras.applications import VGG16, ResNet50, VGG19
from keras.optimizers import Adam
from sklearn.utils import class_weight
```

```
from IPython.display import SVG, Image
```

4 EDA

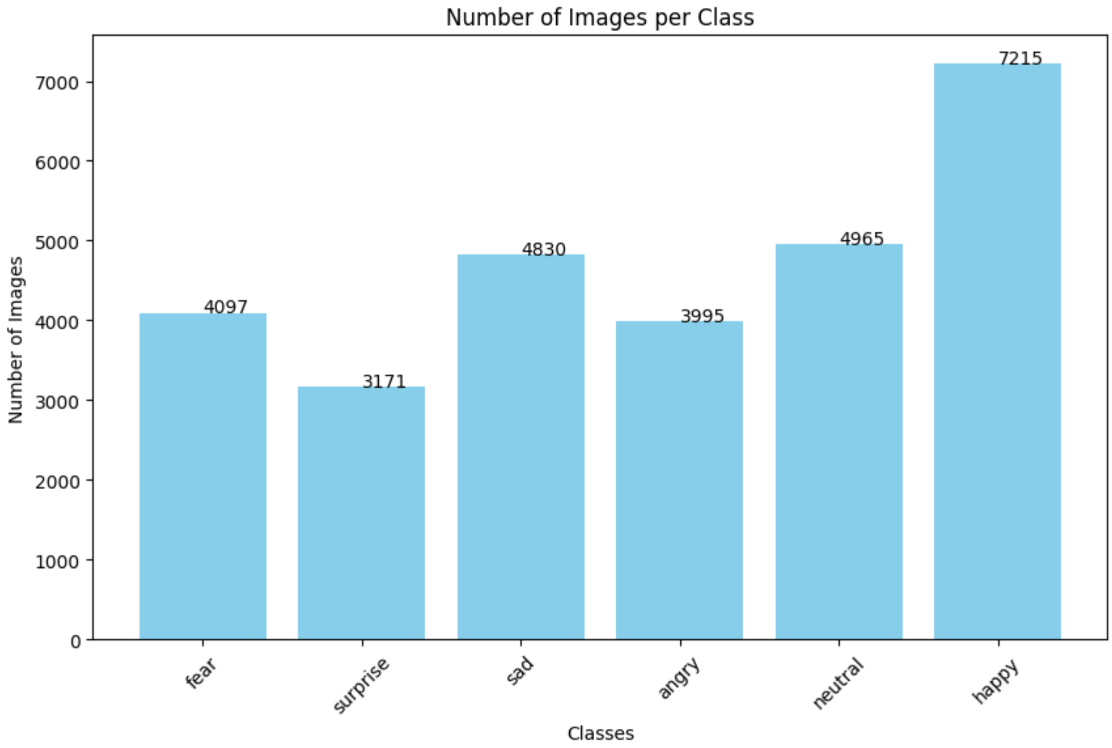
4.1 Displaying Images

[ ]:
```
folder_path = r"data/"
expression = 'happy'
picture_size = 48

plt.figure(figsize= (12,12))
for i in range(1, 10, 1):
    plt.subplot(3,3,i)
    img = load_img(folder_path+"train/"+expression+"/"+
                  os.listdir(folder_path + "train/" + expression)[i],␣
 ↪target_size=(picture_size, picture_size)
                  )
    plt.imshow(img)
plt.show()
```

4.2 The number of images in each class.

[ ]:
```
classes = os.listdir(folder_path + "train/")
class_counts = {cls: len(os.listdir(folder_path + "train/" + cls)) for cls in classes}
plt.figure(figsize=(10, 6))
bars = plt.bar(class_counts.keys(), class_counts.values(), color='skyblue')

for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2.0, yval, int(yval))
plt.xlabel('Classes')
plt.ylabel('Number of Images')
```

```
plt.title('Number of Images per Class')
plt.xticks(rotation=45)
plt.show()
```



5 Training and Validation Data

[ ]:
```
# Define data augmentation and preprocessing for training
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

# Define preprocessing for validation
val_datagen = ImageDataGenerator(rescale=1./255)

# Load training data
train_generator = train_datagen.flow_from_directory(
    folder_path+"train",
    target_size=(48, 48),
    batch_size=32,
```

```
    color_mode='grayscale',
    class_mode='categorical'
)

# Load validation data
val_generator = val_datagen.flow_from_directory(
    folder_path+"test",
    target_size=(48, 48),
    batch_size=32,
    color_mode='grayscale',
    class_mode='categorical'
)

# Assuming y_train is available from your training generator
y_train = train_generator.classes

# Compute class weights
class_weights = class_weight.compute_class_weight('balanced', classes=np.
 ↪unique(y_train), y=y_train)
class_weights = dict(enumerate(class_weights))
```

```
Found 28273 images belonging to 6 classes.
Found 7067 images belonging to 6 classes.
```

[ ]:
```
# Visualize augmented images
img_path = os.path.join(folder_path, "train", classes[0], os.listdir(os.path.
 ↪join(folder_path, "train", classes[0]))[0])
img = load_img(img_path, target_size=(48, 48), color_mode='grayscale')
img_array = img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)

augmented_images = train_datagen.flow(img_array, batch_size=1)

plt.figure(figsize=(12, 12))
for i in range(3):
    plt.subplot(3, 3, i+1)
    batch = augmented_images.next()
    img_augmented = batch[0]
    plt.imshow(img_augmented.squeeze(), cmap='gray')
    plt.axis('off')
plt.tight_layout()
plt.show()
```

6 Model Building

6.1 Model from Scratch

6.1.1 Model Architecture

[ ]:
```
model= Sequential()

# 1st conv
model.add(Conv2D(64, (3,3), padding='same', input_shape=(48,48,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# 2nd conv
model.add(Conv2D(128, (5,5), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))


# 3rd conv
model.add(Conv2D(512, (3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# 4th conv
model.add(Conv2D(512, (3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
```

```
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# flatten
model.add(Flatten())

# 1st dense layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

# 2nd dense layer
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

#output layer
model.add(Dense(6, activation='softmax'))
```

6.1.2 Compile the Model

[ ]:
```
opt= Adam(learning_rate=0.0005)
model.compile(optimizer= opt, loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 48, 48, 64)        640

 batch_normalization (Batch  (None, 48, 48, 64)        256
 Normalization)

 activation (Activation)     (None, 48, 48, 64)        0

 max_pooling2d (MaxPooling2  (None, 24, 24, 64)        0
 D)

 dropout (Dropout)           (None, 24, 24, 64)        0

 conv2d_1 (Conv2D)           (None, 24, 24, 128)       204928

 batch_normalization_1 (Bat  (None, 24, 24, 128)       512
```

```
chNormalization)

activation_1 (Activation)   (None, 24, 24, 128)       0

max_pooling2d_1 (MaxPoolin  (None, 12, 12, 128)       0
g2D)

dropout_1 (Dropout)         (None, 12, 12, 128)       0

conv2d_2 (Conv2D)           (None, 12, 12, 512)       590336

batch_normalization_2 (Bat  (None, 12, 12, 512)       2048
chNormalization)

activation_2 (Activation)   (None, 12, 12, 512)       0

max_pooling2d_2 (MaxPoolin  (None, 6, 6, 512)         0
g2D)

dropout_2 (Dropout)         (None, 6, 6, 512)         0

conv2d_3 (Conv2D)           (None, 6, 6, 512)         2359808

batch_normalization_3 (Bat  (None, 6, 6, 512)         2048
chNormalization)

activation_3 (Activation)   (None, 6, 6, 512)         0

max_pooling2d_3 (MaxPoolin  (None, 3, 3, 512)         0
g2D)

dropout_3 (Dropout)         (None, 3, 3, 512)         0

flatten (Flatten)           (None, 4608)              0

dense (Dense)               (None, 256)               1179904

batch_normalization_4 (Bat  (None, 256)               1024
chNormalization)

activation_4 (Activation)   (None, 256)               0

dropout_4 (Dropout)         (None, 256)               0

dense_1 (Dense)             (None, 512)               131584

batch_normalization_5 (Bat  (None, 512)               2048
chNormalization)
```

```
 activation_5 (Activation)   (None, 512)               0

 dropout_5 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 6)                 3078

=================================================================
Total params: 4478214 (17.08 MB)
Trainable params: 4474246 (17.07 MB)
Non-trainable params: 3968 (15.50 KB)
_________________________________________________________________
```

6.1.3 Train the Model

[ ]:
```python
# Early stopping callback
early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=10,
    restore_best_weights=True
)

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=100,
    validation_data=val_generator,
    validation_steps=val_generator.samples // val_generator.batch_size,
    verbose=1,
    class_weight=class_weights,
    callbacks=[early_stopping]
)
```

```
Epoch 1/100
883/883 [==============================] - 32s 26ms/step - loss: 1.8082 -
accuracy: 0.2713 - val_loss: 1.6187 - val_accuracy: 0.3415
Epoch 2/100
883/883 [==============================] - 22s 25ms/step - loss: 1.5369 -
accuracy: 0.3938 - val_loss: 1.3703 - val_accuracy: 0.4625
Epoch 3/100
883/883 [==============================] - 22s 25ms/step - loss: 1.3891 -
accuracy: 0.4595 - val_loss: 1.2240 - val_accuracy: 0.5189
Epoch 4/100
883/883 [==============================] - 22s 25ms/step - loss: 1.3069 -
accuracy: 0.4948 - val_loss: 1.1728 - val_accuracy: 0.5430
Epoch 5/100
883/883 [==============================] - 22s 25ms/step - loss: 1.2576 -
accuracy: 0.5183 - val_loss: 1.1711 - val_accuracy: 0.5511
```

```
Epoch 6/100
883/883 [==============================] - 22s 25ms/step - loss: 1.2293 -
accuracy: 0.5316 - val_loss: 1.2066 - val_accuracy: 0.5287
Epoch 7/100
883/883 [==============================] - 22s 25ms/step - loss: 1.1979 -
accuracy: 0.5431 - val_loss: 1.1164 - val_accuracy: 0.5688
Epoch 8/100
883/883 [==============================] - 22s 25ms/step - loss: 1.1719 -
accuracy: 0.5578 - val_loss: 1.0810 - val_accuracy: 0.5817
Epoch 9/100
883/883 [==============================] - 22s 25ms/step - loss: 1.1556 -
accuracy: 0.5607 - val_loss: 1.0740 - val_accuracy: 0.5942
Epoch 10/100
883/883 [==============================] - 22s 25ms/step - loss: 1.1356 -
accuracy: 0.5719 - val_loss: 1.1775 - val_accuracy: 0.5399
Epoch 11/100
883/883 [==============================] - 22s 25ms/step - loss: 1.1164 -
accuracy: 0.5786 - val_loss: 1.0766 - val_accuracy: 0.5892
Epoch 12/100
883/883 [==============================] - 22s 25ms/step - loss: 1.0958 -
accuracy: 0.5837 - val_loss: 1.1272 - val_accuracy: 0.5666
Epoch 13/100
883/883 [==============================] - 22s 25ms/step - loss: 1.0811 -
accuracy: 0.5940 - val_loss: 1.1540 - val_accuracy: 0.5567
Epoch 14/100
883/883 [==============================] - 22s 25ms/step - loss: 1.0615 -
accuracy: 0.5986 - val_loss: 1.0518 - val_accuracy: 0.5849
Epoch 15/100
883/883 [==============================] - 22s 25ms/step - loss: 1.0558 -
accuracy: 0.6038 - val_loss: 1.0066 - val_accuracy: 0.6187
Epoch 16/100
883/883 [==============================] - 22s 25ms/step - loss: 1.0390 -
accuracy: 0.6096 - val_loss: 1.0458 - val_accuracy: 0.5938
Epoch 17/100
883/883 [==============================] - 22s 25ms/step - loss: 1.0239 -
accuracy: 0.6140 - val_loss: 1.0603 - val_accuracy: 0.5908
Epoch 18/100
883/883 [==============================] - 22s 25ms/step - loss: 1.0117 -
accuracy: 0.6209 - val_loss: 1.0132 - val_accuracy: 0.6206
Epoch 19/100
883/883 [==============================] - 22s 25ms/step - loss: 1.0042 -
accuracy: 0.6236 - val_loss: 1.0464 - val_accuracy: 0.6055
Epoch 20/100
883/883 [==============================] - 22s 25ms/step - loss: 0.9865 -
accuracy: 0.6282 - val_loss: 1.1002 - val_accuracy: 0.5776
Epoch 21/100
883/883 [==============================] - 22s 25ms/step - loss: 0.9785 -
accuracy: 0.6325 - val_loss: 1.0582 - val_accuracy: 0.5986
```

```
Epoch 22/100
883/883 [==============================] - 22s 25ms/step - loss: 0.9695 -
accuracy: 0.6391 - val_loss: 0.9652 - val_accuracy: 0.6408
Epoch 23/100
883/883 [==============================] - 22s 25ms/step - loss: 0.9594 -
accuracy: 0.6429 - val_loss: 1.0450 - val_accuracy: 0.6006
Epoch 24/100
883/883 [==============================] - 22s 25ms/step - loss: 0.9489 -
accuracy: 0.6467 - val_loss: 1.0076 - val_accuracy: 0.6214
Epoch 25/100
883/883 [==============================] - 22s 25ms/step - loss: 0.9344 -
accuracy: 0.6478 - val_loss: 0.9511 - val_accuracy: 0.6489
Epoch 26/100
883/883 [==============================] - 22s 25ms/step - loss: 0.9256 -
accuracy: 0.6541 - val_loss: 1.3178 - val_accuracy: 0.5224
Epoch 27/100
883/883 [==============================] - 22s 25ms/step - loss: 0.9187 -
accuracy: 0.6559 - val_loss: 0.9648 - val_accuracy: 0.6438
Epoch 28/100
883/883 [==============================] - 22s 25ms/step - loss: 0.9071 -
accuracy: 0.6623 - val_loss: 0.9584 - val_accuracy: 0.6486
Epoch 29/100
883/883 [==============================] - 22s 25ms/step - loss: 0.8951 -
accuracy: 0.6689 - val_loss: 1.0871 - val_accuracy: 0.5933
Epoch 30/100
883/883 [==============================] - 22s 25ms/step - loss: 0.8789 -
accuracy: 0.6721 - val_loss: 1.0138 - val_accuracy: 0.6247
Epoch 31/100
883/883 [==============================] - 23s 25ms/step - loss: 0.8725 -
accuracy: 0.6755 - val_loss: 1.0411 - val_accuracy: 0.6097
Epoch 32/100
883/883 [==============================] - 22s 25ms/step - loss: 0.8634 -
accuracy: 0.6790 - val_loss: 1.0681 - val_accuracy: 0.6078
Epoch 33/100
883/883 [==============================] - 22s 25ms/step - loss: 0.8575 -
accuracy: 0.6807 - val_loss: 1.0928 - val_accuracy: 0.5884
Epoch 34/100
883/883 [==============================] - 22s 25ms/step - loss: 0.8515 -
accuracy: 0.6842 - val_loss: 1.0314 - val_accuracy: 0.6173
Epoch 35/100
883/883 [==============================] - 22s 25ms/step - loss: 0.8307 -
accuracy: 0.6910 - val_loss: 0.9362 - val_accuracy: 0.6582
Epoch 36/100
883/883 [==============================] - 22s 25ms/step - loss: 0.8283 -
accuracy: 0.6933 - val_loss: 1.0914 - val_accuracy: 0.5933
Epoch 37/100
883/883 [==============================] - 22s 25ms/step - loss: 0.8267 -
accuracy: 0.6931 - val_loss: 0.9885 - val_accuracy: 0.6422
```

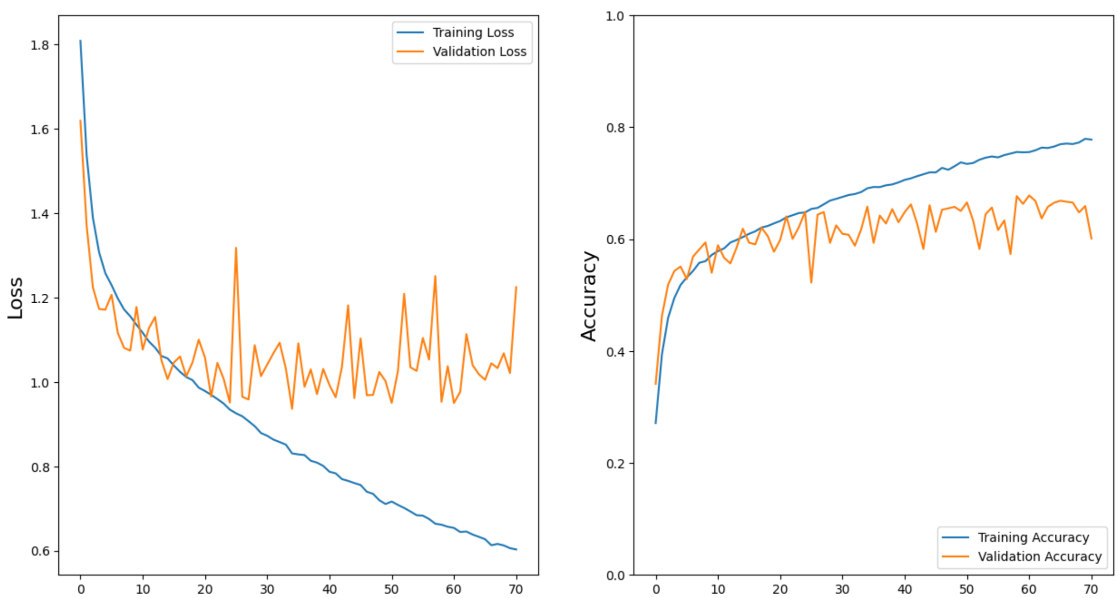
```
Epoch 38/100
883/883 [==============================] - 22s 25ms/step - loss: 0.8132 -
accuracy: 0.6963 - val_loss: 1.0297 - val_accuracy: 0.6280
Epoch 39/100
883/883 [==============================] - 22s 25ms/step - loss: 0.8088 -
accuracy: 0.6979 - val_loss: 0.9712 - val_accuracy: 0.6537
Epoch 40/100
883/883 [==============================] - 22s 25ms/step - loss: 0.8011 -
accuracy: 0.7015 - val_loss: 1.0307 - val_accuracy: 0.6303
Epoch 41/100
883/883 [==============================] - 22s 25ms/step - loss: 0.7872 -
accuracy: 0.7060 - val_loss: 0.9925 - val_accuracy: 0.6484
Epoch 42/100
883/883 [==============================] - 22s 25ms/step - loss: 0.7834 -
accuracy: 0.7088 - val_loss: 0.9637 - val_accuracy: 0.6622
Epoch 43/100
883/883 [==============================] - 22s 25ms/step - loss: 0.7698 -
accuracy: 0.7127 - val_loss: 1.0348 - val_accuracy: 0.6277
Epoch 44/100
883/883 [==============================] - 22s 25ms/step - loss: 0.7655 -
accuracy: 0.7160 - val_loss: 1.1815 - val_accuracy: 0.5825
Epoch 45/100
883/883 [==============================] - 22s 25ms/step - loss: 0.7604 -
accuracy: 0.7195 - val_loss: 0.9616 - val_accuracy: 0.6607
Epoch 46/100
883/883 [==============================] - 22s 25ms/step - loss: 0.7558 -
accuracy: 0.7193 - val_loss: 1.1030 - val_accuracy: 0.6131
Epoch 47/100
883/883 [==============================] - 22s 25ms/step - loss: 0.7398 -
accuracy: 0.7275 - val_loss: 0.9685 - val_accuracy: 0.6527
Epoch 48/100
883/883 [==============================] - 22s 25ms/step - loss: 0.7349 -
accuracy: 0.7241 - val_loss: 0.9694 - val_accuracy: 0.6551
Epoch 49/100
883/883 [==============================] - 22s 25ms/step - loss: 0.7198 -
accuracy: 0.7303 - val_loss: 1.0236 - val_accuracy: 0.6578
Epoch 50/100
883/883 [==============================] - 22s 25ms/step - loss: 0.7108 -
accuracy: 0.7373 - val_loss: 1.0021 - val_accuracy: 0.6504
Epoch 51/100
883/883 [==============================] - 22s 25ms/step - loss: 0.7166 -
accuracy: 0.7345 - val_loss: 0.9504 - val_accuracy: 0.6656
Epoch 52/100
883/883 [==============================] - 22s 25ms/step - loss: 0.7087 -
accuracy: 0.7362 - val_loss: 1.0261 - val_accuracy: 0.6330
Epoch 53/100
883/883 [==============================] - 22s 25ms/step - loss: 0.7014 -
accuracy: 0.7420 - val_loss: 1.2088 - val_accuracy: 0.5825
```

```
Epoch 54/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6931 -
accuracy: 0.7457 - val_loss: 1.0348 - val_accuracy: 0.6446
Epoch 55/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6843 -
accuracy: 0.7478 - val_loss: 1.0261 - val_accuracy: 0.6565
Epoch 56/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6833 -
accuracy: 0.7462 - val_loss: 1.1042 - val_accuracy: 0.6163
Epoch 57/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6753 -
accuracy: 0.7503 - val_loss: 1.0528 - val_accuracy: 0.6335
Epoch 58/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6641 -
accuracy: 0.7531 - val_loss: 1.2509 - val_accuracy: 0.5734
Epoch 59/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6617 -
accuracy: 0.7558 - val_loss: 0.9528 - val_accuracy: 0.6770
Epoch 60/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6569 -
accuracy: 0.7552 - val_loss: 1.0372 - val_accuracy: 0.6631
Epoch 61/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6541 -
accuracy: 0.7556 - val_loss: 0.9502 - val_accuracy: 0.6781
Epoch 62/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6445 -
accuracy: 0.7589 - val_loss: 0.9763 - val_accuracy: 0.6683
Epoch 63/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6456 -
accuracy: 0.7637 - val_loss: 1.1133 - val_accuracy: 0.6371
Epoch 64/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6384 -
accuracy: 0.7630 - val_loss: 1.0396 - val_accuracy: 0.6580
Epoch 65/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6330 -
accuracy: 0.7656 - val_loss: 1.0182 - val_accuracy: 0.6651
Epoch 66/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6273 -
accuracy: 0.7697 - val_loss: 1.0050 - val_accuracy: 0.6687
Epoch 67/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6131 -
accuracy: 0.7709 - val_loss: 1.0442 - val_accuracy: 0.6669
Epoch 68/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6162 -
accuracy: 0.7701 - val_loss: 1.0328 - val_accuracy: 0.6655
Epoch 69/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6126 -
accuracy: 0.7728 - val_loss: 1.0679 - val_accuracy: 0.6479
```

```
Epoch 70/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6061 -
accuracy: 0.7794 - val_loss: 1.0210 - val_accuracy: 0.6592
Epoch 71/100
883/883 [==============================] - 22s 25ms/step - loss: 0.6031 -
accuracy: 0.7780 - val_loss: 1.2246 - val_accuracy: 0.6013
```

6.1.4 Plotting Accuracy & Loss

[ ]:
```python
plt.figure(figsize=(15,8))
plt.subplot(1, 2, 1)
plt.ylabel('Loss', fontsize=16)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend(loc='upper right')

plt.subplot(1, 2, 2)
plt.ylabel('Accuracy', fontsize=16)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylim(0, 1)
plt.show()
```



6.1.5 Evaluate the Model

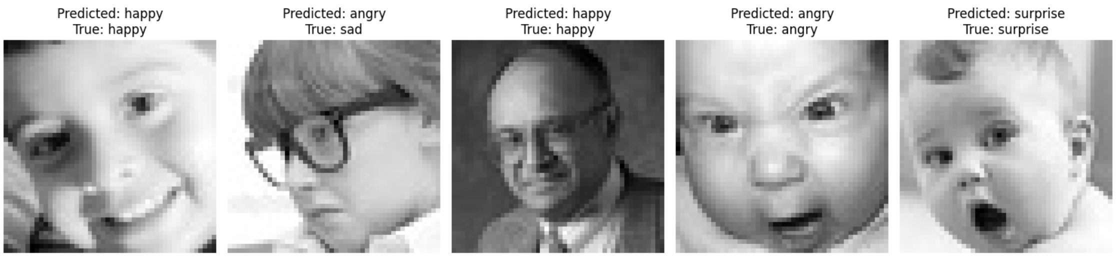
```
[ ]: test_loss, test_acc = model.evaluate(val_generator, steps=val_generator.samples␣
      ↪// val_generator.batch_size)
     print(f"Validation accuracy: {test_acc}")
```

```
220/220 [==============================] - 2s 11ms/step - loss: 0.9549 -
accuracy: 0.6776
Validation accuracy: 0.6775568127632141
```

6.1.6 Make Predictions

```
[ ]: val_images, val_labels = next(val_generator)

     # Select 5 images
     selected_images = val_images[:5]
     selected_labels = val_labels[:5]

     # Make predictions
     predictions = model.predict(selected_images)
     predicted_classes = np.argmax(predictions, axis=1)
     true_classes = np.argmax(selected_labels, axis=1)
     class_labels = list(val_generator.class_indices.keys())

     # Print the predicted and true labels for each image
     fig, axes = plt.subplots(1, len(selected_images), figsize=(15, 5))
     for i, ax in enumerate(axes):
         ax.imshow(selected_images[i].reshape(48, 48), cmap='gray')
         ax.set_title(f"Predicted: {class_labels[predicted_classes[i]]}\nTrue:␣
      ↪{class_labels[true_classes[i]]}")
         ax.axis('off')

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

```
1/1 [==============================] - 0s 23ms/step
```



6.1.7 Save Model

[ ]:
```
model.save('my_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

6.1.8 Load Model

[ ]:
```
#model = tf.keras.models.load_model('my_model.h5')
```

7 Pre-trained Models

7.1 VGG16

[ ]:
```
# Load the VGG16 model
base_model = VGG16(weights='imagenet', include_top=False,␣
 ↪input_tensor=Input(shape=(48, 48, 3)))

# Convert grayscale to RGB
input_layer = Input(shape=(48, 48, 1))
x = Conv2D(3, (1, 1))(input_layer)

x = base_model(x)
x = GlobalAveragePooling2D()(x)

x = Dense(128)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.25)(x)

x = Dense(256)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.25)(x)

output_layer = Dense(6, activation='softmax')(x)

vgg_model = Model(inputs=input_layer, outputs=output_layer)

vgg_model.summary()

# fine-tuning
for layer in base_model.layers[-20:]:
```

```
    layer.trainable = True

vgg_model.compile(optimizer=Adam(learning_rate=1e-5),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])

early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True )

# training
history_VGG = vgg_model.fit(
    train_generator,
    epochs=50,
    validation_data=val_generator,
    class_weight=class_weights,
    callbacks=[early_stopping]
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 1s 0us/step
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 48, 48, 1)]       0

 conv2d_4 (Conv2D)           (None, 48, 48, 3)         6

 vgg16 (Functional)          (None, 1, 1, 512)         14714688

 global_average_pooling2d (  (None, 512)               0
 GlobalAveragePooling2D)

 dense_3 (Dense)             (None, 128)               65664

 batch_normalization_6 (Bat  (None, 128)               512
 chNormalization)

 activation_6 (Activation)   (None, 128)               0

 dropout_6 (Dropout)         (None, 128)               0

 dense_4 (Dense)             (None, 256)               33024

 batch_normalization_7 (Bat  (None, 256)               1024
```

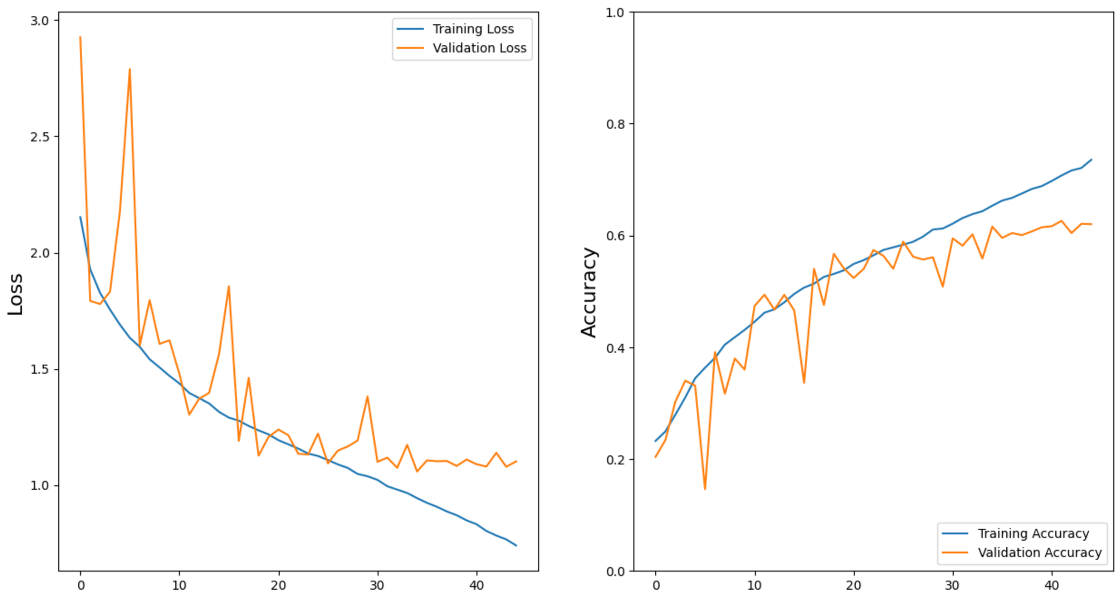
```
 chNormalization)

 activation_7 (Activation)   (None, 256)              0

 dropout_7 (Dropout)         (None, 256)              0

 dense_5 (Dense)             (None, 6)                1542

=================================================================
Total params: 14816460 (56.52 MB)
Trainable params: 14815692 (56.52 MB)
Non-trainable params: 768 (3.00 KB)
_________________________________________________________________
Epoch 1/50
884/884 [==============================] - 32s 27ms/step - loss: 2.1524 -
accuracy: 0.2325 - val_loss: 2.9261 - val_accuracy: 0.2039
Epoch 2/50
884/884 [==============================] - 23s 26ms/step - loss: 1.9279 -
accuracy: 0.2496 - val_loss: 1.7919 - val_accuracy: 0.2343
Epoch 3/50
884/884 [==============================] - 24s 27ms/step - loss: 1.8261 -
accuracy: 0.2795 - val_loss: 1.7787 - val_accuracy: 0.3027
Epoch 4/50
884/884 [==============================] - 23s 26ms/step - loss: 1.7545 -
accuracy: 0.3102 - val_loss: 1.8319 - val_accuracy: 0.3402
Epoch 5/50
884/884 [==============================] - 23s 26ms/step - loss: 1.6900 -
accuracy: 0.3446 - val_loss: 2.1781 - val_accuracy: 0.3314
Epoch 6/50
884/884 [==============================] - 23s 26ms/step - loss: 1.6331 -
accuracy: 0.3635 - val_loss: 2.7895 - val_accuracy: 0.1462
Epoch 7/50
884/884 [==============================] - 23s 26ms/step - loss: 1.5945 -
accuracy: 0.3811 - val_loss: 1.6006 - val_accuracy: 0.3914
Epoch 8/50
884/884 [==============================] - 23s 26ms/step - loss: 1.5409 -
accuracy: 0.4048 - val_loss: 1.7951 - val_accuracy: 0.3170
Epoch 9/50
884/884 [==============================] - 23s 26ms/step - loss: 1.5059 -
accuracy: 0.4180 - val_loss: 1.6073 - val_accuracy: 0.3799
Epoch 10/50
884/884 [==============================] - 23s 25ms/step - loss: 1.4695 -
accuracy: 0.4312 - val_loss: 1.6224 - val_accuracy: 0.3601
Epoch 11/50
884/884 [==============================] - 23s 26ms/step - loss: 1.4371 -
accuracy: 0.4458 - val_loss: 1.4786 - val_accuracy: 0.4740
Epoch 12/50
884/884 [==============================] - 23s 26ms/step - loss: 1.3960 -
```

```
accuracy: 0.4619 - val_loss: 1.3030 - val_accuracy: 0.4940
Epoch 13/50
884/884 [==============================] - 23s 25ms/step - loss: 1.3736 -
accuracy: 0.4679 - val_loss: 1.3713 - val_accuracy: 0.4675
Epoch 14/50
884/884 [==============================] - 22s 25ms/step - loss: 1.3511 -
accuracy: 0.4804 - val_loss: 1.3964 - val_accuracy: 0.4938
Epoch 15/50
884/884 [==============================] - 23s 26ms/step - loss: 1.3148 -
accuracy: 0.4955 - val_loss: 1.5621 - val_accuracy: 0.4660
Epoch 16/50
884/884 [==============================] - 23s 26ms/step - loss: 1.2899 -
accuracy: 0.5068 - val_loss: 1.8552 - val_accuracy: 0.3365
Epoch 17/50
884/884 [==============================] - 23s 26ms/step - loss: 1.2769 -
accuracy: 0.5138 - val_loss: 1.1898 - val_accuracy: 0.5404
Epoch 18/50
884/884 [==============================] - 23s 26ms/step - loss: 1.2542 -
accuracy: 0.5260 - val_loss: 1.4609 - val_accuracy: 0.4756
Epoch 19/50
884/884 [==============================] - 23s 26ms/step - loss: 1.2352 -
accuracy: 0.5312 - val_loss: 1.1263 - val_accuracy: 0.5671
Epoch 20/50
884/884 [==============================] - 23s 25ms/step - loss: 1.2181 -
accuracy: 0.5375 - val_loss: 1.2063 - val_accuracy: 0.5414
Epoch 21/50
884/884 [==============================] - 23s 25ms/step - loss: 1.1929 -
accuracy: 0.5491 - val_loss: 1.2387 - val_accuracy: 0.5240
Epoch 22/50
884/884 [==============================] - 23s 25ms/step - loss: 1.1753 -
accuracy: 0.5558 - val_loss: 1.2145 - val_accuracy: 0.5400
Epoch 23/50
884/884 [==============================] - 23s 26ms/step - loss: 1.1569 -
accuracy: 0.5645 - val_loss: 1.1347 - val_accuracy: 0.5739
Epoch 24/50
884/884 [==============================] - 23s 26ms/step - loss: 1.1354 -
accuracy: 0.5740 - val_loss: 1.1311 - val_accuracy: 0.5637
Epoch 25/50
884/884 [==============================] - 22s 25ms/step - loss: 1.1248 -
accuracy: 0.5788 - val_loss: 1.2213 - val_accuracy: 0.5405
Epoch 26/50
884/884 [==============================] - 23s 26ms/step - loss: 1.1076 -
accuracy: 0.5835 - val_loss: 1.0932 - val_accuracy: 0.5889
Epoch 27/50
884/884 [==============================] - 23s 25ms/step - loss: 1.0888 -
accuracy: 0.5887 - val_loss: 1.1482 - val_accuracy: 0.5622
Epoch 28/50
884/884 [==============================] - 23s 26ms/step - loss: 1.0735 -
```

```
accuracy: 0.5979 - val_loss: 1.1658 - val_accuracy: 0.5570
Epoch 29/50
884/884 [==============================] - 23s 26ms/step - loss: 1.0477 -
accuracy: 0.6106 - val_loss: 1.1917 - val_accuracy: 0.5611
Epoch 30/50
884/884 [==============================] - 23s 26ms/step - loss: 1.0377 -
accuracy: 0.6125 - val_loss: 1.3812 - val_accuracy: 0.5086
Epoch 31/50
884/884 [==============================] - 23s 26ms/step - loss: 1.0223 -
accuracy: 0.6211 - val_loss: 1.0999 - val_accuracy: 0.5949
Epoch 32/50
884/884 [==============================] - 23s 26ms/step - loss: 0.9947 -
accuracy: 0.6312 - val_loss: 1.1180 - val_accuracy: 0.5817
Epoch 33/50
884/884 [==============================] - 23s 26ms/step - loss: 0.9803 -
accuracy: 0.6382 - val_loss: 1.0740 - val_accuracy: 0.6021
Epoch 34/50
884/884 [==============================] - 23s 26ms/step - loss: 0.9658 -
accuracy: 0.6434 - val_loss: 1.1730 - val_accuracy: 0.5588
Epoch 35/50
884/884 [==============================] - 23s 26ms/step - loss: 0.9436 -
accuracy: 0.6533 - val_loss: 1.0580 - val_accuracy: 0.6158
Epoch 36/50
884/884 [==============================] - 23s 26ms/step - loss: 0.9234 -
accuracy: 0.6624 - val_loss: 1.1061 - val_accuracy: 0.5957
Epoch 37/50
884/884 [==============================] - 23s 26ms/step - loss: 0.9064 -
accuracy: 0.6673 - val_loss: 1.1023 - val_accuracy: 0.6044
Epoch 38/50
884/884 [==============================] - 23s 26ms/step - loss: 0.8866 -
accuracy: 0.6752 - val_loss: 1.1035 - val_accuracy: 0.6007
Epoch 39/50
884/884 [==============================] - 23s 26ms/step - loss: 0.8700 -
accuracy: 0.6833 - val_loss: 1.0821 - val_accuracy: 0.6073
Epoch 40/50
884/884 [==============================] - 23s 26ms/step - loss: 0.8481 -
accuracy: 0.6884 - val_loss: 1.1097 - val_accuracy: 0.6147
Epoch 41/50
884/884 [==============================] - 23s 26ms/step - loss: 0.8313 -
accuracy: 0.6974 - val_loss: 1.0896 - val_accuracy: 0.6165
Epoch 42/50
884/884 [==============================] - 23s 26ms/step - loss: 0.8024 -
accuracy: 0.7075 - val_loss: 1.0795 - val_accuracy: 0.6261
Epoch 43/50
884/884 [==============================] - 23s 26ms/step - loss: 0.7827 -
accuracy: 0.7162 - val_loss: 1.1388 - val_accuracy: 0.6044
Epoch 44/50
884/884 [==============================] - 23s 26ms/step - loss: 0.7663 -
```

```
accuracy: 0.7208 - val_loss: 1.0787 - val_accuracy: 0.6209
Epoch 45/50
884/884 [==============================] - 23s 26ms/step - loss: 0.7399 -
accuracy: 0.7354 - val_loss: 1.1016 - val_accuracy: 0.6202
```

[ ]:
```
plt.figure(figsize=(15,8))
plt.subplot(1, 2, 1)
plt.ylabel('Loss', fontsize=16)
plt.plot(history_VGG.history['loss'], label='Training Loss')
plt.plot(history_VGG.history['val_loss'], label='Validation Loss')
plt.legend(loc='upper right')

plt.subplot(1, 2, 2)
plt.ylabel('Accuracy', fontsize=16)
plt.plot(history_VGG.history['accuracy'], label='Training Accuracy')
plt.plot(history_VGG.history['val_accuracy'], label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylim(0, 1)
plt.show()
```



[ ]:
```
model.save('VGG16_model.h5')
```

[ ]:
```
test_loss, test_acc = vgg_model.evaluate(val_generator, steps=val_generator.
 ↪samples // val_generator.batch_size)
print(f"Validation accuracy: {test_acc}")
```

```
220/220 [==============================] - 2s 10ms/step - loss: 1.3434 -
accuracy: 0.6438
```

```
Validation accuracy: 0.643750011920929
```

```
[ ]: # Make predictions
predictions = vgg_model.predict(selected_images)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(selected_labels, axis=1)
class_labels = list(val_generator.class_indices.keys())

# Print the predicted and true labels for each image in the same row
fig, axes = plt.subplots(1, len(selected_images), figsize=(15, 5))
for i, ax in enumerate(axes):
    ax.imshow(selected_images[i].reshape(48, 48), cmap='gray')
    ax.set_title(f"Predicted: {class_labels[predicted_classes[i]]}\nTrue: {class_labels[true_classes[i]]}")
    ax.axis('off')

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

```
1/1 [==============================] - 0s 26ms/step
```

7.2 ResNet50

```
[ ]: # Load the ResNet50 model
base_model = ResNet50(weights='imagenet', include_top=False, input_tensor=Input(shape=(48, 48, 3)))

input_layer = Input(shape=(48, 48, 1))
x = Conv2D(3, (1, 1))(input_layer)

x = base_model(x)
x = GlobalAveragePooling2D()(x)

x = Dense(128)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.25)(x)
```

```python
x = Dense(256)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.25)(x)

output_layer = Dense(6, activation='softmax')(x)

resnet50_model = Model(inputs=input_layer, outputs=output_layer)

for layer in base_model.layers[-10:]:
    layer.trainable = True

# Compile
resnet50_model.compile(optimizer=Adam(learning_rate=1e-5),␣
↪loss='categorical_crossentropy', metrics=['accuracy'])

resnet50_model.summary()

# Early Stopping Callback
early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=10,
    restore_best_weights=True
)

# Training
history_resnet50 = resnet50_model.fit(
    train_generator,
    epochs=50,
    validation_data=val_generator,
    class_weight=class_weights,
    callbacks=[early_stopping]
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 1s 0us/step
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 48, 48, 1)]       0

 conv2d_5 (Conv2D)           (None, 48, 48, 3)         6

 resnet50 (Functional)       (None, 2, 2, 2048)        23587712
```

```
 global_average_pooling2d_1  (None, 2048)              0
  (GlobalAveragePooling2D)

 dense_6 (Dense)             (None, 128)               262272

 batch_normalization_8 (Bat  (None, 128)               512
 chNormalization)

 activation_8 (Activation)   (None, 128)               0

 dropout_8 (Dropout)         (None, 128)               0

 dense_7 (Dense)             (None, 256)               33024

 batch_normalization_9 (Bat  (None, 256)               1024
 chNormalization)

 activation_9 (Activation)   (None, 256)               0

 dropout_9 (Dropout)         (None, 256)               0

 dense_8 (Dense)             (None, 6)                 1542

=================================================================
Total params: 23886092 (91.12 MB)
Trainable params: 23832204 (90.91 MB)
Non-trainable params: 53888 (210.50 KB)
_________________________________________________________________
Epoch 1/50
884/884 [==============================] - 80s 52ms/step - loss: 2.0816 -
accuracy: 0.1823 - val_loss: 32.5903 - val_accuracy: 0.1885
Epoch 2/50
884/884 [==============================] - 41s 47ms/step - loss: 1.8886 -
accuracy: 0.2442 - val_loss: 1.8034 - val_accuracy: 0.3266
Epoch 3/50
884/884 [==============================] - 41s 47ms/step - loss: 1.7735 -
accuracy: 0.2902 - val_loss: 1.6728 - val_accuracy: 0.3622
Epoch 4/50
884/884 [==============================] - 41s 47ms/step - loss: 1.6851 -
accuracy: 0.3261 - val_loss: 1.5751 - val_accuracy: 0.3811
Epoch 5/50
884/884 [==============================] - 41s 47ms/step - loss: 1.6225 -
accuracy: 0.3488 - val_loss: 1.5237 - val_accuracy: 0.4054
Epoch 6/50
884/884 [==============================] - 41s 47ms/step - loss: 1.5616 -
accuracy: 0.3753 - val_loss: 1.4663 - val_accuracy: 0.4272
Epoch 7/50
```

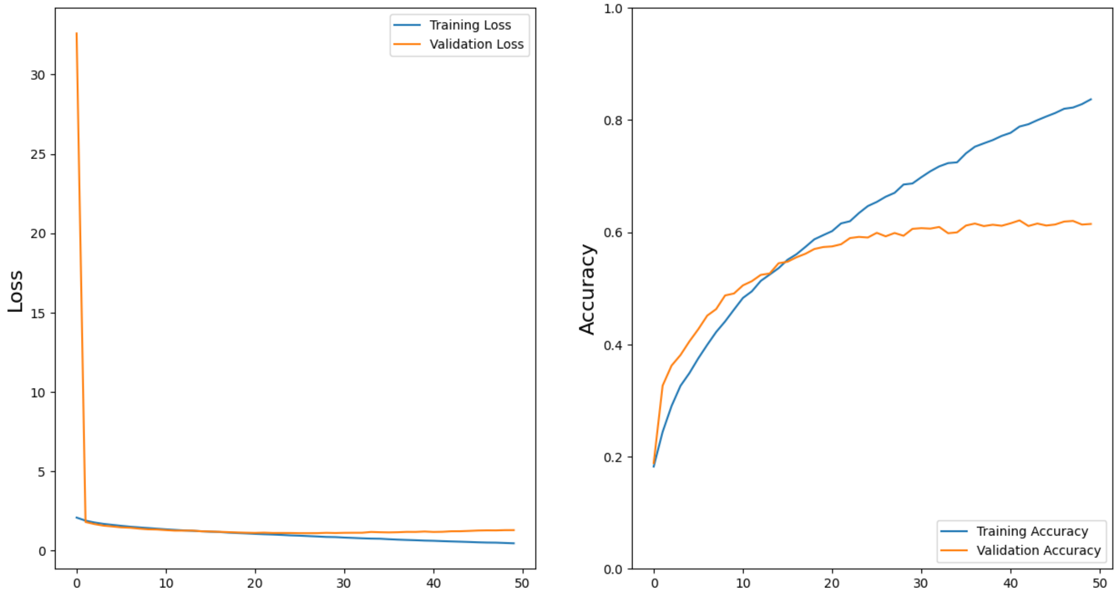
```
884/884 [==============================] - 41s 47ms/step - loss: 1.5080 -
accuracy: 0.3993 - val_loss: 1.4411 - val_accuracy: 0.4514
Epoch 8/50
884/884 [==============================] - 41s 46ms/step - loss: 1.4646 -
accuracy: 0.4222 - val_loss: 1.3856 - val_accuracy: 0.4630
Epoch 9/50
884/884 [==============================] - 41s 47ms/step - loss: 1.4237 -
accuracy: 0.4410 - val_loss: 1.3417 - val_accuracy: 0.4873
Epoch 10/50
884/884 [==============================] - 42s 47ms/step - loss: 1.3836 -
accuracy: 0.4622 - val_loss: 1.3294 - val_accuracy: 0.4909
Epoch 11/50
884/884 [==============================] - 42s 47ms/step - loss: 1.3421 -
accuracy: 0.4829 - val_loss: 1.2929 - val_accuracy: 0.5053
Epoch 12/50
884/884 [==============================] - 42s 47ms/step - loss: 1.3065 -
accuracy: 0.4947 - val_loss: 1.2603 - val_accuracy: 0.5128
Epoch 13/50
884/884 [==============================] - 43s 48ms/step - loss: 1.2747 -
accuracy: 0.5133 - val_loss: 1.2603 - val_accuracy: 0.5240
Epoch 14/50
884/884 [==============================] - 43s 48ms/step - loss: 1.2481 -
accuracy: 0.5246 - val_loss: 1.2614 - val_accuracy: 0.5265
Epoch 15/50
884/884 [==============================] - 42s 48ms/step - loss: 1.2200 -
accuracy: 0.5360 - val_loss: 1.2160 - val_accuracy: 0.5448
Epoch 16/50
884/884 [==============================] - 43s 48ms/step - loss: 1.1898 -
accuracy: 0.5509 - val_loss: 1.2049 - val_accuracy: 0.5475
Epoch 17/50
884/884 [==============================] - 42s 48ms/step - loss: 1.1747 -
accuracy: 0.5608 - val_loss: 1.1861 - val_accuracy: 0.5554
Epoch 18/50
884/884 [==============================] - 42s 47ms/step - loss: 1.1342 -
accuracy: 0.5738 - val_loss: 1.1659 - val_accuracy: 0.5616
Epoch 19/50
884/884 [==============================] - 42s 48ms/step - loss: 1.1082 -
accuracy: 0.5874 - val_loss: 1.1448 - val_accuracy: 0.5701
Epoch 20/50
884/884 [==============================] - 42s 48ms/step - loss: 1.0840 -
accuracy: 0.5948 - val_loss: 1.1291 - val_accuracy: 0.5737
Epoch 21/50
884/884 [==============================] - 42s 48ms/step - loss: 1.0551 -
accuracy: 0.6021 - val_loss: 1.1182 - val_accuracy: 0.5748
Epoch 22/50
884/884 [==============================] - 42s 48ms/step - loss: 1.0329 -
accuracy: 0.6158 - val_loss: 1.1335 - val_accuracy: 0.5786
Epoch 23/50
```

```
884/884 [==============================] - 42s 48ms/step - loss: 1.0140 -
accuracy: 0.6197 - val_loss: 1.1128 - val_accuracy: 0.5896
Epoch 24/50
884/884 [==============================] - 42s 47ms/step - loss: 0.9889 -
accuracy: 0.6342 - val_loss: 1.1137 - val_accuracy: 0.5918
Epoch 25/50
884/884 [==============================] - 41s 47ms/step - loss: 0.9599 -
accuracy: 0.6467 - val_loss: 1.1108 - val_accuracy: 0.5906
Epoch 26/50
884/884 [==============================] - 41s 47ms/step - loss: 0.9423 -
accuracy: 0.6541 - val_loss: 1.1007 - val_accuracy: 0.5990
Epoch 27/50
884/884 [==============================] - 41s 46ms/step - loss: 0.9161 -
accuracy: 0.6634 - val_loss: 1.1003 - val_accuracy: 0.5928
Epoch 28/50
884/884 [==============================] - 42s 47ms/step - loss: 0.8905 -
accuracy: 0.6703 - val_loss: 1.1024 - val_accuracy: 0.5987
Epoch 29/50
884/884 [==============================] - 42s 47ms/step - loss: 0.8627 -
accuracy: 0.6851 - val_loss: 1.1220 - val_accuracy: 0.5937
Epoch 30/50
884/884 [==============================] - 42s 48ms/step - loss: 0.8527 -
accuracy: 0.6870 - val_loss: 1.1094 - val_accuracy: 0.6061
Epoch 31/50
884/884 [==============================] - 42s 48ms/step - loss: 0.8230 -
accuracy: 0.6982 - val_loss: 1.1226 - val_accuracy: 0.6073
Epoch 32/50
884/884 [==============================] - 42s 48ms/step - loss: 0.8010 -
accuracy: 0.7087 - val_loss: 1.1261 - val_accuracy: 0.6065
Epoch 33/50
884/884 [==============================] - 42s 48ms/step - loss: 0.7759 -
accuracy: 0.7174 - val_loss: 1.1264 - val_accuracy: 0.6095
Epoch 34/50
884/884 [==============================] - 42s 48ms/step - loss: 0.7580 -
accuracy: 0.7233 - val_loss: 1.1769 - val_accuracy: 0.5983
Epoch 35/50
884/884 [==============================] - 43s 48ms/step - loss: 0.7498 -
accuracy: 0.7247 - val_loss: 1.1593 - val_accuracy: 0.5998
Epoch 36/50
884/884 [==============================] - 43s 48ms/step - loss: 0.7179 -
accuracy: 0.7407 - val_loss: 1.1479 - val_accuracy: 0.6120
Epoch 37/50
884/884 [==============================] - 42s 48ms/step - loss: 0.6924 -
accuracy: 0.7526 - val_loss: 1.1594 - val_accuracy: 0.6155
Epoch 38/50
884/884 [==============================] - 42s 48ms/step - loss: 0.6722 -
accuracy: 0.7585 - val_loss: 1.1817 - val_accuracy: 0.6110
Epoch 39/50
```

```
884/884 [==============================] - 42s 48ms/step - loss: 0.6544 -
accuracy: 0.7644 - val_loss: 1.1789 - val_accuracy: 0.6134
Epoch 40/50
884/884 [==============================] - 42s 48ms/step - loss: 0.6322 -
accuracy: 0.7718 - val_loss: 1.2041 - val_accuracy: 0.6116
Epoch 41/50
884/884 [==============================] - 43s 48ms/step - loss: 0.6198 -
accuracy: 0.7773 - val_loss: 1.1772 - val_accuracy: 0.6160
Epoch 42/50
884/884 [==============================] - 41s 47ms/step - loss: 0.5978 -
accuracy: 0.7885 - val_loss: 1.1869 - val_accuracy: 0.6212
Epoch 43/50
884/884 [==============================] - 41s 46ms/step - loss: 0.5794 -
accuracy: 0.7926 - val_loss: 1.2193 - val_accuracy: 0.6112
Epoch 44/50
884/884 [==============================] - 41s 46ms/step - loss: 0.5630 -
accuracy: 0.7998 - val_loss: 1.2216 - val_accuracy: 0.6155
Epoch 45/50
884/884 [==============================] - 41s 46ms/step - loss: 0.5448 -
accuracy: 0.8064 - val_loss: 1.2423 - val_accuracy: 0.6120
Epoch 46/50
884/884 [==============================] - 41s 46ms/step - loss: 0.5235 -
accuracy: 0.8126 - val_loss: 1.2667 - val_accuracy: 0.6138
Epoch 47/50
884/884 [==============================] - 42s 47ms/step - loss: 0.5086 -
accuracy: 0.8202 - val_loss: 1.2767 - val_accuracy: 0.6191
Epoch 48/50
884/884 [==============================] - 41s 47ms/step - loss: 0.5023 -
accuracy: 0.8224 - val_loss: 1.2740 - val_accuracy: 0.6202
Epoch 49/50
884/884 [==============================] - 41s 47ms/step - loss: 0.4818 -
accuracy: 0.8285 - val_loss: 1.2900 - val_accuracy: 0.6137
Epoch 50/50
884/884 [==============================] - 42s 47ms/step - loss: 0.4614 -
accuracy: 0.8371 - val_loss: 1.2924 - val_accuracy: 0.6148
```

[ ]:
```python
# Plot Accuracy & Loss
plt.figure(figsize=(15,8))
plt.subplot(1, 2, 1)
plt.ylabel('Loss', fontsize=16)
plt.plot(history_resnet50.history['loss'], label='Training Loss')
plt.plot(history_resnet50.history['val_loss'], label='Validation Loss')
plt.legend(loc='upper right')

plt.subplot(1, 2, 2)
plt.ylabel('Accuracy', fontsize=16)
plt.plot(history_resnet50.history['accuracy'], label='Training Accuracy')
```

```
plt.plot(history_resnet50.history['val_accuracy'], label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylim(0, 1)
plt.show()
```



[ ]:
```
# Save model
model.save('ResNet50_model.h5')
```

[ ]:
```
# Evaluate model
test_loss, test_acc = resnet50_model.evaluate(val_generator,␣
↪steps=val_generator.samples // val_generator.batch_size)
print(f"Validation accuracy: {test_acc}")
```

```
220/220 [==============================] - 3s 11ms/step - loss: 1.2924 -
accuracy: 0.6148
Validation accuracy: 0.6147727370262146
```

[ ]:
```
# Make predictions
predictions = resnet50_model.predict(selected_images)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(selected_labels, axis=1)
class_labels = list(val_generator.class_indices.keys())

# Print the predicted and true labels for each image
fig, axes = plt.subplots(1, len(selected_images), figsize=(15, 5))
for i, ax in enumerate(axes):
    ax.imshow(selected_images[i].reshape(48, 48), cmap='gray')
```

```
    ax.set_title(f"Predicted: {class_labels[predicted_classes[i]]}\nTrue:␣
 ↪{class_labels[true_classes[i]]}")
    ax.axis('off')

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

```
1/1 [==============================] - 0s 28ms/step
```


```
[ ]:
```