Machine Learning Major Project : COMP8220

Name : Karthik Varma Keerthipati

Student ID: 46272496

Contact : karthikvarma.keerthipati@students.mq.edu.au

In this project we are working on image dataset to predict the labels which are emotions of faces in images. There are seven emotions(labels) angry, disgusted, fearful, happy, sad, surprised, neutral. In order to predict the labels, we are using machine learning models. Some Conventional models and Deep learning models. we are able to build this based on the refrences form the lecture notes and Colab transfer learning tutorial.

We are having datasets for training images, training labels, public test images and private test images. We are going to split the training images dataset into train and validation set/ dev set which is used to test the performance of the model.

We are importing the required libraries to build the models.

The Conventional models:

Support Vector Machine classifier

Logistic Regression

Random Forest Classifier

Support vector classifier is a supervised machine learning algorithm and we are using it for classification task. This algorithm is based on support vectors(data points) and depends on kernal for the seperation of the classes.

Logistic regression is used to predict the probability of target variable. It is based on the logistic function.

Random forest classifier is used for classification tasks by creating decision trees on randomly selected data and depends on the feature importance. The model having more decision trees will give the best results.

In [1]:
```python
# import the libraries
import keras
import warnings
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
from os.path import join
from sklearn.svm import SVC
from google.colab import drive
from keras.models import Model
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
from keras.models import Sequential
from matplotlib.pyplot import imshow
from keras.preprocessing import image
from sklearn.metrics import accuracy_score
from tensorflow.keras import layers, models
from sklearn.metrics import confusion_matrix
from keras.layers import Conv2D, MaxPooling2D
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.imagenet_utils import preprocess_input
from keras.layers import Dense, Dropout, Flatten, Activation, BatchNormalization
from tensorflow.keras.preprocessing.image import load_img, img_to_array, array_to_img
```

In [2]:
```python
# mount drive and directory
drive.mount('/content/drive/', force_remount = True)

image_size = 48
EMOTIONS = ['angry', 'disgusted', 'fearful','happy', 'sad', 'surprised', 'neutral']
dataset_directory = '/content/drive/My Drive/ML Project'
```

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

The image size is 48 and the data is in numpy array format. The images are greyscale which is having a single channel.

In [3]:
```python
# load the data
images_train = np.load(join(dataset_directory, 'Training_data_images.npy'))
labels_train = np.load(join(dataset_directory, 'Training_data_labels.npy'))

images_test_public = np.load(join(dataset_directory, 'PublicTest_data_images.npy'))
images_test_private = np.load(join(dataset_directory, 'PrivateTest_data_images.npy'))

#reshaping the data
images_train = images_train.reshape([-1, image_size, image_size])
images_test_public = images_test_public.reshape([-1, image_size, image_size])
images_test_private = images_test_private.reshape([-1, image_size, image_size])

# print the shape
print(images_train.shape)
print(labels_train.shape)
print(images_test_public.shape)
print(images_test_private.shape)
```

```
(9328, 48, 48)
(9328,)
(1136, 48, 48)
(1153, 48, 48)
```

In [4]:
```python
print(images_train[0:1])
```

```
[[[0.25098039 0.24705882 0.2745098  ... 0.16078431 0.2        0.21960784]
  [0.22745098 0.19215686 0.20784314 ... 0.14509804 0.16470588 0.18823529]
  [0.12941176 0.14901961 0.16862745 ... 0.17254902 0.21568627 0.20784314]
  ...
  [0.14117647 0.19215686 0.23137255 ... 0.19607843 0.22352941 0.25882353]
  [0.12156863 0.12941176 0.18431373 ... 0.2627451  0.26666667 0.23137255]
  [0.14901961 0.09019608 0.09803922 ... 0.37254902 0.26666667 0.21568627]]]
```

In [5]:
```python
# print sample data with emotion angry
sample = 0
print(images_train[sample])
print(labels_train[sample])
print(EMOTIONS[labels_train[sample]])
```

```
[[0.25098039 0.24705882 0.2745098  ... 0.16078431 0.2        0.21960784]
 [0.22745098 0.19215686 0.20784314 ... 0.14509804 0.16470588 0.18823529]
 [0.12941176 0.14901961 0.16862745 ... 0.17254902 0.21568627 0.20784314]
 ...
 [0.14117647 0.19215686 0.23137255 ... 0.19607843 0.22352941 0.25882353]
 [0.12156863 0.12941176 0.18431373 ... 0.2627451  0.26666667 0.23137255]
 [0.14901961 0.09019608 0.09803922 ... 0.37254902 0.26666667 0.21568627]]
0
angry
```

In [6]:
```python
# plot the images with emotions
plt.figure(figsize=(10,10))
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(images_train[i], cmap='gray')
    plt.xlabel( EMOTIONS[ labels_train[i] ] )
plt.show()
```

Preprocessing the data and extracting the features.

Splitting the data into train and validation set and can be used to test the model performance.

In [7]:
```python
# split the data into train and val
images_train, images_val = train_test_split(images_train, test_size=0.2, random_state=42)
labels_train, labels_val = train_test_split(labels_train, test_size=0.2, random_state=42)
```

In [8]:
```python
# convert the values
def convert_values (count,method):
    result=[]
    for i in range(count):
        Output= method[i].ravel()
        result.append(Output)
    return result

image_train = convert_values(7462, images_train)
image_val = convert_values(1866, images_val)
image_test_public = convert_values(1136, images_test_public)
image_test_private =convert_values(1153,images_test_private)
```

In [9]:
```python
img_train = np.array(image_train)
img_val = np.array(image_val)
img_test_public = np.array(image_test_public)
img_test_private =np.array(image_test_private)
```

In [10]:
```python
# print the input data shape
print(img_train.shape)
print(img_val.shape)
print(labels_train.shape)
print(labels_val.shape)
print(img_test_public.shape)
print(img_test_private.shape)
```

```
(7462, 2304)
(1866, 2304)
(7462,)
(1866,)
(1136, 2304)
(1153, 2304)
```

SVM Classifier

The final model for the Kaggle submission that produced the best-performing predictions and got an accuracy of 0.54313 on the public test images set by using SVC with a Radial Basis Function(rbf) kernel.

We are using grid search to get the best parameters and estimators for the model with the data.

In [ ]:
```python
# define parameter
param_grid_search = {'C': [0.1, 1, 10, 100], 'gamma': [0.01, 0.001, 0.0001], 'kernel': ['rbf', 'poly']}


grid_search = GridSearchCV(SVC(), param_grid_search, refit = True, verbose = 2)

# fit the model for grid search
grid_search.fit(img_train, labels_train)

# print best parameter
print(grid_search.best_params_)

# print best estimator
print(grid_search.best_estimator_)
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
[CV] C=0.1, gamma=0.01, kernel=rbf ..................................
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV] ................... C=0.1, gamma=0.01, kernel=rbf, total= 3.8min
[CV] C=0.1, gamma=0.01, kernel=rbf ..................................
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  3.8min remaining:    0.0s
[CV] ................... C=0.1, gamma=0.01, kernel=rbf, total= 3.8min
[CV] C=0.1, gamma=0.01, kernel=rbf ..................................
[CV] ................... C=0.1, gamma=0.01, kernel=rbf, total= 3.7min
[CV] C=0.1, gamma=0.01, kernel=rbf ..................................
[CV] ................... C=0.1, gamma=0.01, kernel=rbf, total= 3.7min
[CV] C=0.1, gamma=0.01, kernel=rbf ..................................
```

It is taking very long time to process the grid search on my computer because of limited resources of my system.

In [11]:
```python
clf = SVC(C = 100, kernel = 'rbf', gamma = 'scale', random_state = 42)

# fit the model on the train data
history = clf.fit(img_train, labels_train)

# predicting on validation data
pred = clf.predict(img_val)

Accuracy = accuracy_score(pred, labels_val)

# print accuracy
print("The accuracy for SVM model:", Accuracy)
```

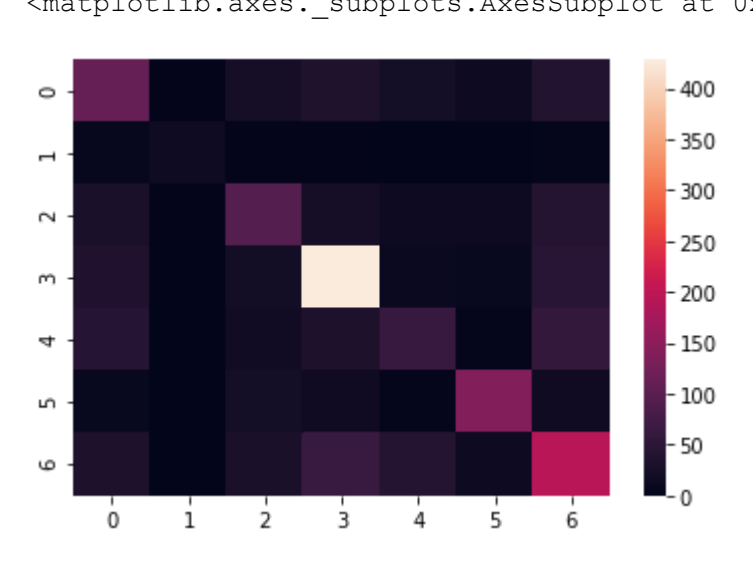
```
The accuracy for SVM model: 0.5621650589496249
```

In [12]:
```python
# confusion matrix
print("The Confusion Matrix for SVM Classifier is: ", "\n", confusion_matrix(labels_val, pred))
```

```
The Confusion Matrix for SVM Classifier is:
 [[113   2  25  34  23  15  40]
 [  8  16   2   2   1   1   4]
 [ 29   2  94  24  15  14  41]
 [ 36   0  21 428  13  10  46]
 [ 42   0  19  32  62   5  57]
 [ 10   0  23  17   5 141  17]
 [ 33   1  29  64  41  14 195]]
```

In [13]:
```python
# heatmap for confusion matrix
sns.heatmap(confusion_matrix(labels_val, pred))
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff529bbb6a0>
```



By observing the above confusion matrix and heatmap we can say which labels are misclassified by the model. The label 0 is classified 113 as correctly but 2 are misclassified as label 1, 25 are misclassified as label 2, 34 as label 3, 23 as label 4, 15 as label 5, 40 as label 6. In the same way, we can say for all the labels.

In [14]:
```python
# predicting the labels for public test images
pred1 = clf.predict(img_test_public)
pred1
```

Out[14]:
```
array([3, 4, 2, ..., 5, 3, 6])
```

In [15]:
```python
# creating a dataframe
pred1_df = pd.DataFrame(pred1, columns = ["Predictions"])
pred1_df
```

Out[15]:

	Predictions
0	3
1	4
2	2
3	2
4	6
...	...
1131	5
1132	0
1133	5
1134	3
1135	6

1136 rows × 1 columns

In [16]:
```python
# save dataframe to csv format
pred1_df.to_csv(r'/content/drive/My Drive/ML Project/46272496_conv.csv', index = True)
```

Logistic Regression

In [17]:
```python
clf = LogisticRegression(random_state = 42)

# fit the train data and labels
clf.fit(img_train, labels_train)

# predicting on validation data
pred = clf.predict(img_val)

Accuracy1 = accuracy_score(pred, labels_val)

# print accuracy
print("The accuracy for Logistic Regression model:", Accuracy1)
```

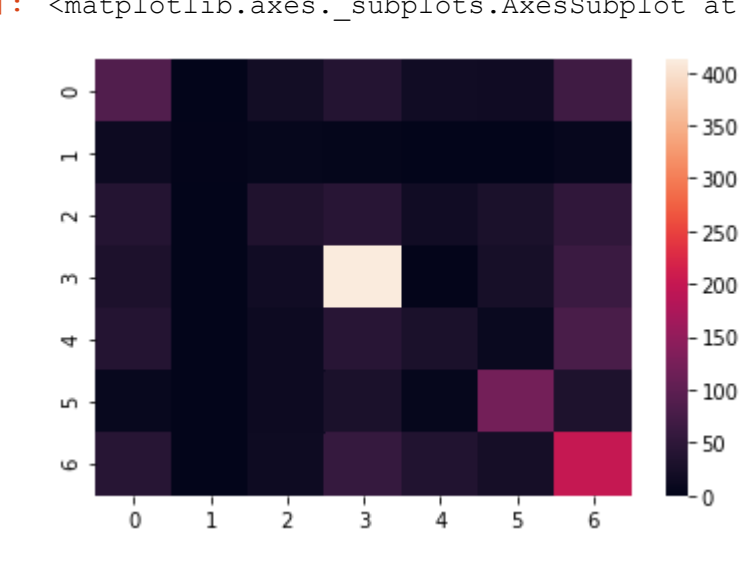
```
The accuracy for Logistic Regression model: 0.4753483386923901
```

In [18]:
```python
# confusion matrix
confusion_matrix(labels_val, pred)
```

Out[18]:
```
array([[ 87,   1,  20,  40,  18,  17,  69],
       [ 13,   2,   5,   4,   1,   1,   8],
       [ 40,   0,  34,  45,  18,  29,  53],
       [ 32,   0,  19, 412,   3,  25,  63],
       [ 40,   0,  14,  44,  30,  10,  79],
       [  9,   0,  13,  30,   6, 122,  33],
       [ 42,   0,  16,  59,  36,  24, 200]])
```

In [19]:
```python
# heatmap for confusion matrix
sns.heatmap(confusion_matrix(labels_val, pred))
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff520285978>
```



By observing the above heatmap and confusion matrix, the label 3 and label 5 are correclty classiffied with very less misclassification.

In [20]:
```python
# predicting the labels for public test images
pred2 = clf.predict(img_test_public)
pred2
```

Out[20]:
```
array([3, 4, 2, ..., 3, 3, 4])
```

RandomForestClassifier

In [21]:
```python
rf = RandomForestClassifier(n_estimators = 500 ,max_depth = 15 ,random_state = 42, bootstrap = True)

# fit the train data and labels
rf.fit(img_train, labels_train)

# predicting on validation data
pred = rf.predict(img_val)

Accuracy1 = accuracy_score(pred, labels_val)
# print accuracy
print("The accuracy for SVM model:", Accuracy1)
```

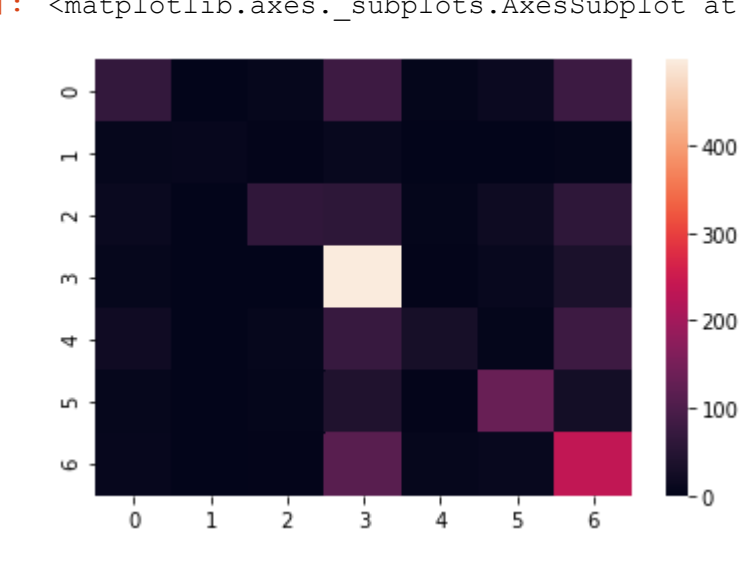
```
The accuracy for SVM model: 0.5562700964630225
```

In [22]:
```python
# confusion matrix
confusion_matrix(labels_val, pred)
```

Out[22]:
```
array([[ 68,   0,   6,  80,   5,  15,  78],
       [  6,   9,   2,  11,   0,   1,   5],
       [ 12,   0,  63,  60,   4,  18,  62],
       [  6,   0,   1, 498,   3,  10,  36],
       [ 21,   0,   7,  73,  31,   4,  81],
       [  7,   0,   4,  41,   2, 133,  26],
       [  9,   0,   3, 113,   6,  10, 236]])
```

In [23]:
```python
# heatmap for confusion matrix
sns.heatmap(confusion_matrix(labels_val, pred))
```

Out[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff5201d4518>
```



By observing the above heatmap and confusion matrix, the label 3 is correclty classiffied with very less misclassification by the model.

In [24]:
```python
# predicting the labels for public test images
pred2 = rf.predict(img_test_public)
pred2
```

Out[24]:
```
array([3, 3, 6, ..., 3, 3, 6])
```

The parameters used for the final model(SVM Classifier) are gamma, C parameter is the regularization parameter and we are taking C as 100 and kernel is rbf. The best result we got out of these three model is from support vector machine classifier. We had selected this model because it is producing slightly better results than the other two models. This may have been because of the overfitting of the model although we tried tuning with hyper parameters.

The accuracy on validation test images for conventional ML models are :

Support Vector Classifier : 0.5621

Logistic Regression : 0.4753

Random Forest Classifier : 0.5562

Deep Learning Models

CNN with two convolutional layers

CNN with more convolutional layers

Transfer Learning to a pretrained network - VGG16

Data Augmentation to a pretrained network

The final model for the Kaggle submission that produced the best performing predictions and got an accuracy of 0.61091 on the public test of images was a fully connected dense model with neural network layers.

The input was the raw data that should be preprocessed to be accepted by the model. AS the images are greyscale, the channel is 1 and we are mentioning it with the help of reshape. There are total 9328 images and 7 labels in the data. The 9328 images will be splitted into train and validation sets. The images are in size of 48. The public test set is containing 1136 images.

In [25]:
```python
# load the data
images_train = np.load(join(dataset_directory, 'Training_data_images.npy'))
labels_train = np.load(join(dataset_directory, 'Training_data_labels.npy'))

images_test_public = np.load(join(dataset_directory, 'PublicTest_data_images.npy'))
images_test_private = np.load(join(dataset_directory, 'PrivateTest_data_images.npy'))

# reshaping the data
images_train = images_train.reshape([-1, image_size, image_size, 1])
images_test_public = images_test_public.reshape([-1, image_size, image_size, 1])
images_test_private = images_test_private.reshape([-1, image_size, image_size, 1])

print(images_train.shape)
print(labels_train.shape)
print(images_test_public.shape)
print(images_test_private.shape)
```

```
(9328, 48, 48, 1)
(9328,)
(1136, 48, 48, 1)
(1153, 48, 48, 1)
```

In [26]:
```python
# split the data into train and validation set
images_train, images_val = train_test_split(images_train, test_size=0.2, random_state=42)
labels_train, labels_val = train_test_split(labels_train, test_size=0.2, random_state=42)
```

In [27]:
```python
# convert from integers to floats
images_train = images_train.astype('float32')
images_val = images_val.astype('float32')
images_test_public = images_test_public.astype('float32')

# normalize to range 0-1
images_train = images_train / 255.0
images_val = images_val / 255.0
images_test_public = images_test_public / 255.0
```

In [28]:
```python
# number of labels
num_classes = len(EMOTIONS)
```

In [29]:
```python
# convert labels to one-hot vectors
labels_train = keras.utils.to_categorical(labels_train, num_classes)
labels_val = keras.utils.to_categorical(labels_val, num_classes)
print(labels_val.shape)
```

```
(1866, 7)
```

In [30]:
```python
# print train data shape
images_train.shape
```

Out[30]:
```
(7462, 48, 48, 1)
```

CNN with two convolutional layers

The model is built with filters from 32 to 64, kernel size is 3, activation function is relu and maxpooling is 2. We are using dropout layer to overcome the overfitting. The dense layer is having 7 classes. Batch normatization is used in data normalization.

In [31]:
```python
# build the network
model = Sequential()
print("Input dimensions: ", images_train.shape[1:])

model.add(Conv2D(32, (3, 3), input_shape = images_train.shape[1:]))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))

model.add(Dense(num_classes))
model.add(Activation('softmax'))
model.summary()
```

```
Input dimensions:  (48, 48, 1)
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 46, 46, 32)        320
_________________________________________________________________
activation (Activation)      (None, 46, 46, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 46, 46, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 23, 23, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 21, 21, 64)        18496
_________________________________________________________________
activation_1 (Activation)    (None, 21, 21, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 21, 21, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 10, 10, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 6400)              0
_________________________________________________________________
dense (Dense)                (None, 256)               1638656
_________________________________________________________________
activation_2 (Activation)    (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 1799
_________________________________________________________________
activation_3 (Activation)    (None, 7)                 0
=================================================================
Total params: 1,659,655
Trainable params: 1,659,463
Non-trainable params: 192
_________________________________________________________________
```

In [32]:
```python
# compile the model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 46, 46, 32)        320
_________________________________________________________________
activation (Activation)      (None, 46, 46, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 46, 46, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 23, 23, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 21, 21, 64)        18496
_________________________________________________________________
activation_1 (Activation)    (None, 21, 21, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 21, 21, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 10, 10, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 6400)              0
_________________________________________________________________
dense (Dense)                (None, 256)               1638656
_________________________________________________________________
activation_2 (Activation)    (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 1799
_________________________________________________________________
activation_3 (Activation)    (None, 7)                 0
=================================================================
Total params: 1,659,655
Trainable params: 1,659,463
Non-trainable params: 192
_________________________________________________________________
```

In [34]:
```python
# fit the train data
history = model.fit(images_train, labels_train,
                        batch_size=500,
                        epochs=30,
                        validation_data=(images_val, labels_val))
```

```
Epoch 1/30
15/15 [==============================] - 0s 24ms/step - loss: 0.0450 - accuracy: 0.9887 - val_loss: 1.9900 - val_accuracy: 0.3215
Epoch 2/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0350 - accuracy: 0.9916 - val_loss: 1.8895 - val_accuracy: 0.4105
Epoch 3/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0324 - accuracy: 0.9928 - val_loss: 1.7145 - val_accuracy: 0.4416
Epoch 4/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0316 - accuracy: 0.9934 - val_loss: 1.7327 - val_accuracy: 0.4314
Epoch 5/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0272 - accuracy: 0.9948 - val_loss: 1.7653 - val_accuracy: 0.4759
Epoch 6/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0271 - accuracy: 0.9948 - val_loss: 1.6503 - val_accuracy: 0.4861
Epoch 7/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0291 - accuracy: 0.9936 - val_loss: 1.6675 - val_accuracy: 0.5021
Epoch 8/30
15/15 [==============================] - 0s 23ms/step - loss: 0.0272 - accuracy: 0.9938 - val_loss: 1.6183 - val_accuracy: 0.5348
Epoch 9/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0243 - accuracy: 0.9952 - val_loss: 1.5937 - val_accuracy: 0.5322
Epoch 10/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0239 - accuracy: 0.9954 - val_loss: 1.6277 - val_accuracy: 0.5627
Epoch 11/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0218 - accuracy: 0.9960 - val_loss: 1.6424 - val_accuracy: 0.5643
Epoch 12/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0191 - accuracy: 0.9965 - val_loss: 1.8770 - val_accuracy: 0.5536
Epoch 13/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0190 - accuracy: 0.9962 - val_loss: 1.8522 - val_accuracy: 0.5809
Epoch 14/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0196 - accuracy: 0.9954 - val_loss: 1.8343 - val_accuracy: 0.5868
Epoch 15/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0200 - accuracy: 0.9952 - val_loss: 1.7788 - val_accuracy: 0.6008
Epoch 16/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0221 - accuracy: 0.9950 - val_loss: 1.7448 - val_accuracy: 0.5959
Epoch 17/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0186 - accuracy: 0.9953 - val_loss: 1.7516 - val_accuracy: 0.6083
Epoch 18/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0184 - accuracy: 0.9952 - val_loss: 1.7784 - val_accuracy: 0.5986
Epoch 19/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0254 - accuracy: 0.9946 - val_loss: 1.9344 - val_accuracy: 0.6083
Epoch 20/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0233 - accuracy: 0.9950 - val_loss: 1.9891 - val_accuracy: 0.6141
Epoch 21/30
15/15 [==============================] - 0s 23ms/step - loss: 0.0244 - accuracy: 0.9940 - val_loss: 2.0619 - val_accuracy: 0.6131
Epoch 22/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0206 - accuracy: 0.9962 - val_loss: 2.1087 - val_accuracy: 0.6206
Epoch 23/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0181 - accuracy: 0.9957 - val_loss: 2.0709 - val_accuracy: 0.6131
Epoch 24/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0188 - accuracy: 0.9961 - val_loss: 2.1934 - val_accuracy: 0.6308
Epoch 25/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0177 - accuracy: 0.9961 - val_loss: 2.2347 - val_accuracy: 0.6088
Epoch 26/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0165 - accuracy: 0.9961 - val_loss: 2.2196 - val_accuracy: 0.6147
Epoch 27/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0168 - accuracy: 0.9961 - val_loss: 2.2741 - val_accuracy: 0.6125
Epoch 28/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0142 - accuracy: 0.9964 - val_loss: 2.3912 - val_accuracy: 0.6206
Epoch 29/30
15/15 [==============================] - 0s 23ms/step - loss: 0.0132 - accuracy: 0.9968 - val_loss: 2.3721 - val_accuracy: 0.5809
Epoch 30/30
15/15 [==============================] - 0s 22ms/step - loss: 0.0170 - accuracy: 0.9961 - val_loss: 2.3925 - val_accuracy: 0.6158
```

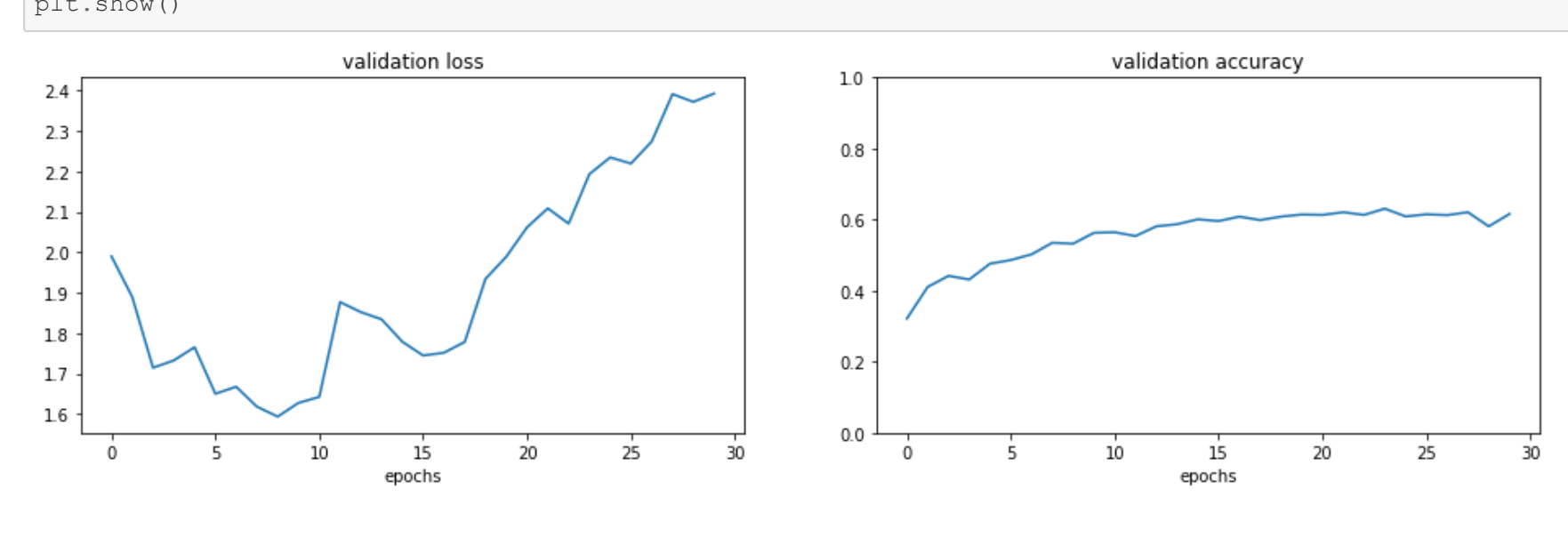
In [35]:
```python
# accuracy
loss, accuracy = model.evaluate(images_val, labels_val, verbose=0)
print('Test loss:', loss)
print('Test accuracy:', accuracy)
```

```
Test loss: 2.3925254344940186
Test accuracy: 0.6157556176185608
```

In [36]:
```python
fig = plt.figure(figsize=(16,4))
ax = fig.add_subplot(121)
ax.plot(history.history["val_loss"])
ax.set_title("validation loss")
ax.set_xlabel("epochs")

ax2 = fig.add_subplot(122)
ax2.plot(history.history["val_accuracy"])
ax2.set_title("validation accuracy")
ax2.set_xlabel("epochs")
ax2.set_ylim(0, 1)

plt.show()
```



Generally the validation loss needs to decrease and the validation accuracy increases.

CNN with more convolutional layers

We are using more network layers with batch noralization and kernal size is 3, activation is relu and also with the dropout layers as 0.5 and padding as same.