In [2]:

```
pip install pickle-mixin
```

```
Requirement already satisfied: pickle-mixin in c:\users\hp info\appdata\loca
l\programs\python\python39\lib\site-packages (1.0.2)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:

```
pip install opencv-python
```

```
Requirement already satisfied: opencv-python in c:\users\hp info\appdata\loc
al\programs\python\python39\lib\site-packages (4.5.4.60)
Requirement already satisfied: numpy>=1.19.3 in c:\users\hp info\appdata\loc
al\programs\python\python39\lib\site-packages (from opencv-python) (1.21.2)
Note: you may need to restart the kernel to use updated packages.
```

In [4]:

```
from os import walk

filename = []

for (dirpath, dirnames, filenames) in walk("C:\\Users\\HP Info\\Documents\\MACS\\Semester 3
    for idx in range(len(filenames)):
        file = filenames[idx].split("_")
        filename.append(file[0])
```

In [15]:

```
#Below code is adapted from https://github.com/soumilshah1995/Smart-Library-to-load-image-D
try:

    import tensorflow as tf
    import cv2
    import os
    import pickle
    import numpy as np
    print("Library Loaded Successfully ..........")
except:
    print("Library not Found ! ")


class MasterImage(object):

    def __init__(self,PATH='', IMAGE_SIZE = 50):
        self.PATH = PATH
        self.IMAGE_SIZE = IMAGE_SIZE

        self.image_data = []
        self.x_data = []
        self.y_data = []
        self.CATEGORIES = []

        # This will get List of categories
        self.list_categories = []

    def get_categories(self):
        for path in os.listdir(self.PATH):
            if '.DS_Store' in path:
                pass
            else:
                self.list_categories.append(path)
        print("Found Categories ",self.list_categories,'\n')
        return self.list_categories

    def Process_Image(self):
        try:
            """
            Return Numpy array of image
            :return: X_Data, Y_Data
            """
            self.CATEGORIES = self.get_categories()
            for categories in self.CATEGORIES:

                train_folder_path = os.path.join(self.PATH, categories)
                class_index = self.CATEGORIES.index(categories)

                for img in os.listdir(train_folder_path):
                    new_path = os.path.join(train_folder_path, img)

                    try:        # if any image is corrupted
                        image_data_temp = cv2.imread(new_path,cv2.IMREAD_GRAYSCALE)
                        image_temp_resize = cv2.resize(image_data_temp,(self.IMAGE_SIZE,sel
                        self.image_data.append([image_temp_resize,class_index])
                    except:
                        pass

            data = np.asanyarray(self.image_data)
```

```
            idx = 0;
            # Iterate over the Data
            for x in data:
                self.x_data.append(x[0])        # Get the X_Data
                self.y_data.append(filename[idx])        # get the label
                idx += 1

            X_Data = np.asarray(self.x_data) / (255.0)      # Normalize Data
            Y_Data = np.asarray(self.y_data)

            # reshape x_Data

            X_Data = X_Data.reshape(-1, self.IMAGE_SIZE, self.IMAGE_SIZE, 1)
            #Y_Data = Y_Data.reshape(-1, self.IMAGE_SIZE, self.IMAGE_SIZE, 1)

            return X_Data, Y_Data
        except:
            print("Failed to run Function Process Image ")

    def pickle_image(self):

        """
        :return: None Creates a Pickle Object of DataSet
        """
        # Call the Function and Get the Data
        X_Data,Y_Data = self.Process_Image()

        # Write the Entire Data into a Pickle File
        pickle_out = open('X_Data','wb')
        pickle.dump(X_Data, pickle_out)
        pickle_out.close()

        # Write the Y Label Data
        pickle_out = open('Y_Data', 'wb')
        pickle.dump(Y_Data, pickle_out)
        pickle_out.close()

        print("Pickled Image Successfully ")
        return X_Data,Y_Data

    def load_dataset(self):

        try:
            # Read the Data from Pickle Object
            X_Temp = open('X_Data','rb')
            X_Data = pickle.load(X_Temp)

            Y_Temp = open('Y_Data','rb')
            Y_Data = pickle.load(Y_Temp)

            print('Reading Dataset from PIckle Object')

            return X_Data,Y_Data

        except:
            print('Could not Found Pickle File ')
            print('Loading File and Dataset  ..........')

            X_Data,Y_Data = self.pickle_image()
            return X_Data,Y_Data
```

```
if __name__ == "__main__":
    path = 'C:\\Users\\HP Info\\Documents\\MACS\\Semester 3\\CSCI6515 - MLBG\\Project\\Data
    a = MasterImage(PATH=path,
                    IMAGE_SIZE=80)

    X_Data,Y_Data = a.load_dataset()
    print(X_Data.shape)
    print(Y_Data.shape)
```

```
Library Loaded Successfully ..........
Reading Dataset from PIckle Object
(4415, 80, 80, 1)
(4415,)
```

In [7]:

```
import csv

with open('fishdata.csv', 'w') as f:
    writer = csv.writer(f)
    writer.writerows(zip(filename, X_Data))
```

In [16]:

```
from sklearn.preprocessing import OneHotEncoder
def prepare_inputs(input):
    ohe = OneHotEncoder(handle_unknown='ignore', sparse=False)
    ohe.fit(input)
    input_enc = ohe.transform(input)
    return input_enc
```

In [17]:

```
Y_Data = Y_Data.reshape((-1,1))
Y_Data = prepare_inputs(Y_Data)
```

In [18]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_Data, Y_Data, test_size=0.3, random_s
```

In [19]:

```
#Data Augmentation
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(rotation_range=8,
                             zoom_range=[0.95, 1.05],
                             height_shift_range=0.10,
                             shear_range=0.15)
```

In [60]:

```
# The below code is adapted from https://www.kaggle.com/toregil/welcome-to-deep-learning-cn
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization
from tensorflow.keras.optimizers import Adam
from keras.callbacks import LearningRateScheduler

model = Sequential()

#Layer 1
model.add(Conv2D(filters = 16, kernel_size = (3, 3), activation='relu',
                 input_shape = (80, 80, 1)))
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

#Layer 2
model.add(Conv2D(filters = 32, kernel_size = (3, 3), activation='relu'))
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(193, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer = Adam(learning_rate=1e-4), metric

annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)

model.summary()

hist = model.fit(datagen.flow(X_train, y_train, batch_size=16),
                              epochs=20, #Increase this when not on Kaggle kernel
                              verbose=2,  #1 for ETA, 0 for silent
                              validation_data=(X_test[:400,:], y_test[:400,:]), #For speed
                              callbacks=[annealer])
```

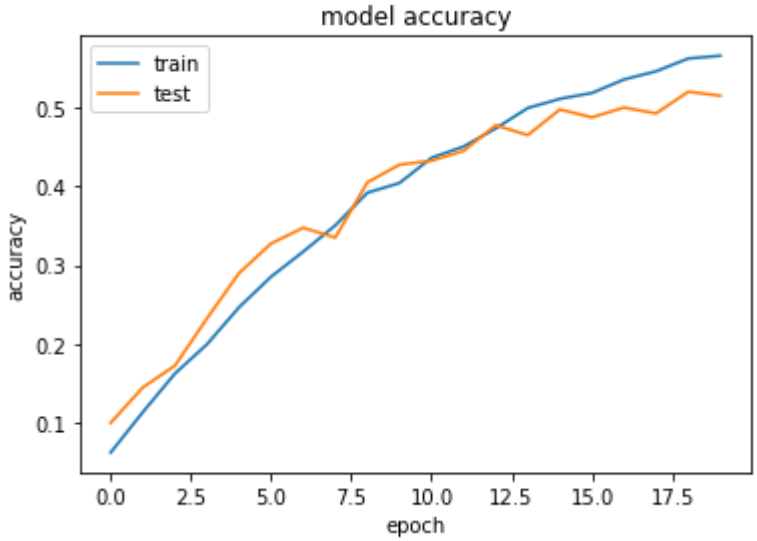
```
Model: "sequential_21"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_51 (Conv2D)          (None, 78, 78, 16)        160

 max_pooling2d_51 (MaxPoolin  (None, 39, 39, 16)       0
 g2D)

 dropout_93 (Dropout)        (None, 39, 39, 16)        0

 conv2d_52 (Conv2D)          (None, 37, 37, 32)        4640

 max_pooling2d_52 (MaxPoolin  (None, 18, 18, 32)       0
 g2D)

 dropout_94 (Dropout)        (None, 18, 18, 32)        0

 flatten_21 (Flatten)        (None, 10368)             0
```

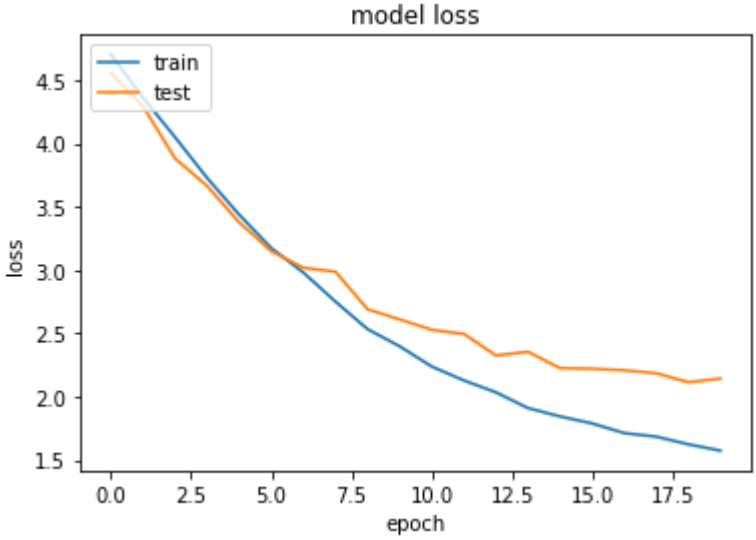
```
 dense_63 (Dense)            (None, 512)               5308928

 dropout_95 (Dropout)        (None, 512)               0

 dense_64 (Dense)            (None, 1024)              525312

 dropout_96 (Dropout)        (None, 1024)              0

 dense_65 (Dense)            (None, 193)               197825

=================================================================
Total params: 6,036,865
Trainable params: 6,036,865
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
194/194 - 16s - loss: 4.7014 - accuracy: 0.0625 - val_loss: 4.5548 - val_acc
uracy: 0.1000 - lr: 0.0010 - 16s/epoch - 83ms/step
Epoch 2/20
194/194 - 16s - loss: 4.3608 - accuracy: 0.1136 - val_loss: 4.2918 - val_acc
uracy: 0.1450 - lr: 9.0000e-04 - 16s/epoch - 85ms/step
Epoch 3/20
194/194 - 16s - loss: 4.0482 - accuracy: 0.1625 - val_loss: 3.8809 - val_acc
uracy: 0.1725 - lr: 8.1000e-04 - 16s/epoch - 83ms/step
Epoch 4/20
194/194 - 16s - loss: 3.7264 - accuracy: 0.1997 - val_loss: 3.6632 - val_acc
uracy: 0.2325 - lr: 7.2900e-04 - 16s/epoch - 84ms/step
Epoch 5/20
194/194 - 14s - loss: 3.4358 - accuracy: 0.2466 - val_loss: 3.3752 - val_acc
uracy: 0.2900 - lr: 6.5610e-04 - 14s/epoch - 75ms/step
Epoch 6/20
194/194 - 14s - loss: 3.1728 - accuracy: 0.2854 - val_loss: 3.1484 - val_acc
uracy: 0.3275 - lr: 5.9049e-04 - 14s/epoch - 74ms/step
Epoch 7/20
194/194 - 14s - loss: 2.9794 - accuracy: 0.3172 - val_loss: 3.0175 - val_acc
uracy: 0.3475 - lr: 5.3144e-04 - 14s/epoch - 71ms/step
Epoch 8/20
194/194 - 17s - loss: 2.7501 - accuracy: 0.3505 - val_loss: 2.9868 - val_acc
uracy: 0.3350 - lr: 4.7830e-04 - 17s/epoch - 87ms/step
Epoch 9/20
194/194 - 14s - loss: 2.5328 - accuracy: 0.3919 - val_loss: 2.6898 - val_acc
uracy: 0.4050 - lr: 4.3047e-04 - 14s/epoch - 72ms/step
Epoch 10/20
194/194 - 15s - loss: 2.3974 - accuracy: 0.4042 - val_loss: 2.6100 - val_acc
uracy: 0.4275 - lr: 3.8742e-04 - 15s/epoch - 76ms/step
Epoch 11/20
194/194 - 14s - loss: 2.2377 - accuracy: 0.4362 - val_loss: 2.5280 - val_acc
uracy: 0.4325 - lr: 3.4868e-04 - 14s/epoch - 71ms/step
Epoch 12/20
194/194 - 16s - loss: 2.1282 - accuracy: 0.4505 - val_loss: 2.4950 - val_acc
uracy: 0.4450 - lr: 3.1381e-04 - 16s/epoch - 81ms/step
Epoch 13/20
194/194 - 14s - loss: 2.0341 - accuracy: 0.4735 - val_loss: 2.3256 - val_acc
uracy: 0.4775 - lr: 2.8243e-04 - 14s/epoch - 73ms/step
Epoch 14/20
194/194 - 15s - loss: 1.9094 - accuracy: 0.4994 - val_loss: 2.3540 - val_acc
uracy: 0.4650 - lr: 2.5419e-04 - 15s/epoch - 75ms/step
Epoch 15/20
194/194 - 14s - loss: 1.8438 - accuracy: 0.5110 - val_loss: 2.2249 - val_acc
uracy: 0.4975 - lr: 2.2877e-04 - 14s/epoch - 74ms/step
```

```
Epoch 16/20
194/194 - 16s - loss: 1.7884 - accuracy: 0.5184 - val_loss: 2.2204 - val_acc
uracy: 0.4875 - lr: 2.0589e-04 - 16s/epoch - 85ms/step
Epoch 17/20
194/194 - 16s - loss: 1.7120 - accuracy: 0.5356 - val_loss: 2.2082 - val_acc
uracy: 0.5000 - lr: 1.8530e-04 - 16s/epoch - 81ms/step
Epoch 18/20
194/194 - 19s - loss: 1.6839 - accuracy: 0.5460 - val_loss: 2.1850 - val_acc
uracy: 0.4925 - lr: 1.6677e-04 - 19s/epoch - 100ms/step
Epoch 19/20
194/194 - 17s - loss: 1.6234 - accuracy: 0.5621 - val_loss: 2.1140 - val_acc
uracy: 0.5200 - lr: 1.5009e-04 - 17s/epoch - 89ms/step
Epoch 20/20
194/194 - 16s - loss: 1.5736 - accuracy: 0.5657 - val_loss: 2.1429 - val_acc
uracy: 0.5150 - lr: 1.3509e-04 - 16s/epoch - 81ms/step
```

In [61]:

```
#Plotting model accuracy and loss
from matplotlib import pyplot as plt
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





In [32]:

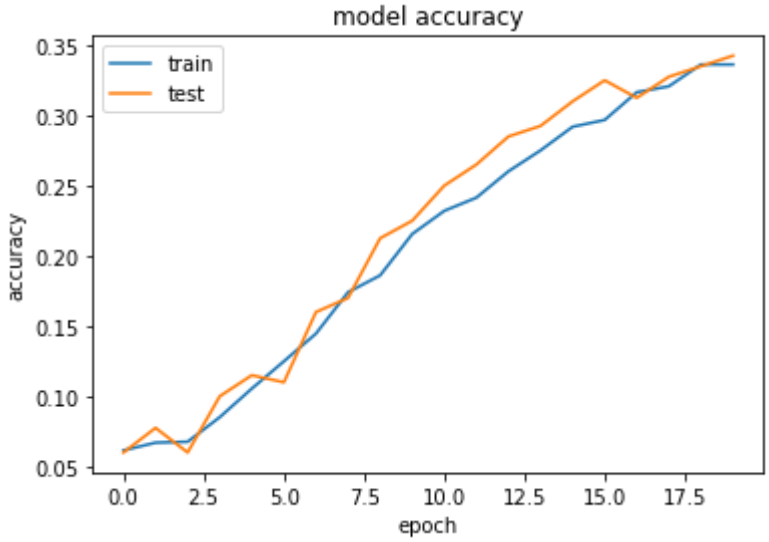
```
# The below code is adapted from https://www.kaggle.com/toregil/welcome-to-deep-learning-cn
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization
from tensorflow.keras.optimizers import Adam
from keras.callbacks import LearningRateScheduler

model = Sequential()

#Layer 1
model.add(Conv2D(filters = 16, kernel_size = (3, 3), activation='relu',
                 input_shape = (80, 80, 1)))
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

#Layer 2
model.add(Conv2D(filters = 32, kernel_size = (3, 3), activation='relu'))
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

#Layer 3
model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation='relu'))
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

#Layer 4
model.add(Conv2D(filters = 128, kernel_size = (3, 3), activation='relu'))
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(193, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer = Adam(learning_rate=1e-4), metric

annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)

model.summary()

hist = model.fit(datagen.flow(X_train, y_train, batch_size=16),
                           epochs=20, #Increase this when not on Kaggle kernel
                           verbose=2,  #1 for ETA, 0 for silent
                           validation_data=(X_test[:400,:], y_test[:400,:]), #For speed
                           callbacks=[annealer])
```

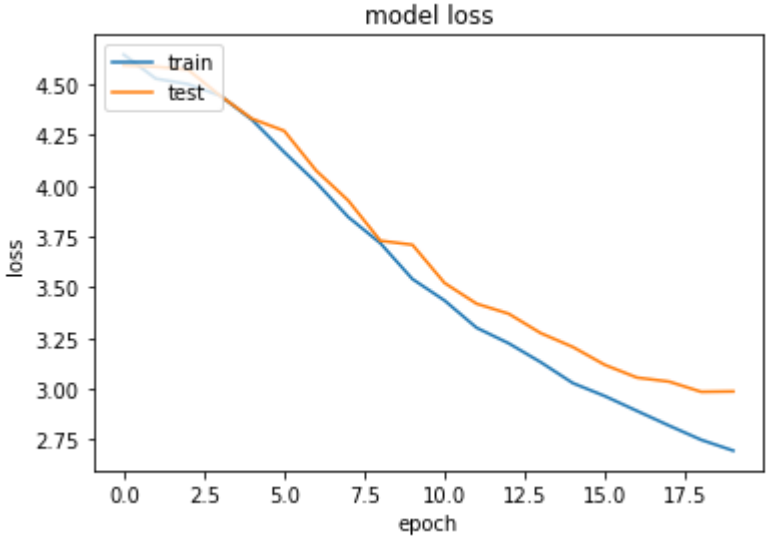
```
accuracy: 0.2925 - lr: 2.5419e-04 - 13s/epoch - 67ms/step
Epoch 15/20
194/194 - 12s - loss: 3.0274 - accuracy: 0.2919 - val_loss: 3.2054 - val_
accuracy: 0.3100 - lr: 2.2877e-04 - 12s/epoch - 59ms/step
Epoch 16/20
194/194 - 11s - loss: 2.9633 - accuracy: 0.2968 - val_loss: 3.1170 - val_
accuracy: 0.3250 - lr: 2.0589e-04 - 11s/epoch - 57ms/step
Epoch 17/20
194/194 - 12s - loss: 2.8903 - accuracy: 0.3165 - val_loss: 3.0545 - val_
accuracy: 0.3125 - lr: 1.8530e-04 - 12s/epoch - 64ms/step
```

```
Epoch 18/20
194/194 - 12s - loss: 2.8180 - accuracy: 0.3207 - val_loss: 3.0348 - val_
accuracy: 0.3275 - lr: 1.6677e-04 - 12s/epoch - 64ms/step
Epoch 19/20
194/194 - 14s - loss: 2.7479 - accuracy: 0.3362 - val_loss: 2.9845 - val_
accuracy: 0.3350 - lr: 1.5009e-04 - 14s/epoch - 71ms/step
Epoch 20/20
194/194 - 18s - loss: 2.6946 - accuracy: 0.3362 - val_loss: 2.9864 - val_
accuracy: 0.3425 - lr: 1.3509e-04 - 18s/epoch - 91ms/step
```

In [33]:

```
#Plotting model accuracy and loss
from matplotlib import pyplot as plt
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





In [34]:

```
# The below code is adapted from https://www.kaggle.com/toregil/welcome-to-deep-learning-cn
#SGD optimizer
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization
from tensorflow.keras.optimizers import SGD
from keras.callbacks import LearningRateScheduler

model = Sequential()

#Layer 1
model.add(Conv2D(filters = 16, kernel_size = (3, 3), activation='relu',
                 input_shape = (80, 80, 1)))
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

#Layer 2
model.add(Conv2D(filters = 32, kernel_size = (3, 3), activation='relu'))
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(193, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer = SGD(learning_rate=1e-4), metrics

annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)

model.summary()

hist = model.fit(datagen.flow(X_train, y_train, batch_size=16),
                              epochs=20, #Increase this when not on Kaggle kernel
                              verbose=2,  #1 for ETA, 0 for silent
                              validation_data=(X_test[:400,:], y_test[:400,:]), #For speed
                              callbacks=[annealer])
```

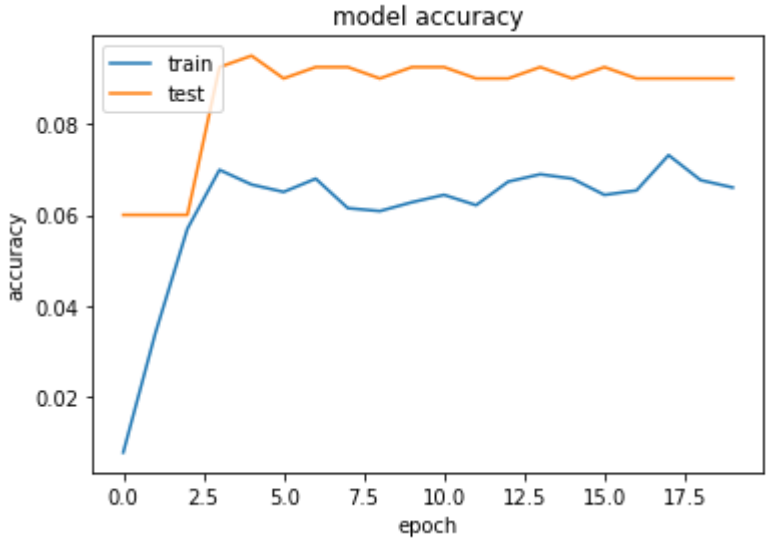
```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_14 (Conv2D)          (None, 78, 78, 16)        160

 max_pooling2d_14 (MaxPoolin  (None, 39, 39, 16)       0
 g2D)

 dropout_26 (Dropout)        (None, 39, 39, 16)        0

 conv2d_15 (Conv2D)          (None, 37, 37, 32)        4640

 max_pooling2d_15 (MaxPoolin  (None, 18, 18, 32)       0
 g2D)

 dropout_27 (Dropout)        (None, 18, 18, 32)        0
```

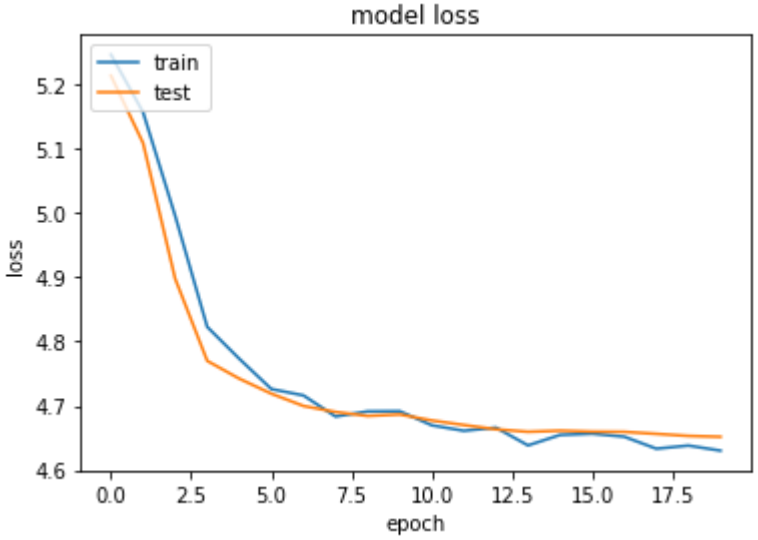
```
 flatten_6 (Flatten)         (None, 10368)             0

 dense_18 (Dense)            (None, 512)               5308928

 dropout_28 (Dropout)        (None, 512)               0

 dense_19 (Dense)            (None, 1024)              525312

 dropout_29 (Dropout)        (None, 1024)              0

 dense_20 (Dense)            (None, 193)               197825

=================================================================
Total params: 6,036,865
Trainable params: 6,036,865
Non-trainable params: 0

_________________________________________________________________
Epoch 1/20
194/194 - 11s - loss: 5.2463 - accuracy: 0.0078 - val_loss: 5.2134 - val_
accuracy: 0.0600 - lr: 0.0010 - 11s/epoch - 54ms/step
Epoch 2/20
194/194 - 11s - loss: 5.1566 - accuracy: 0.0340 - val_loss: 5.1087 - val_
accuracy: 0.0600 - lr: 9.0000e-04 - 11s/epoch - 56ms/step
Epoch 3/20
194/194 - 12s - loss: 4.9957 - accuracy: 0.0570 - val_loss: 4.8978 - val_
accuracy: 0.0600 - lr: 8.1000e-04 - 12s/epoch - 60ms/step
Epoch 4/20
194/194 - 12s - loss: 4.8228 - accuracy: 0.0699 - val_loss: 4.7699 - val_
accuracy: 0.0925 - lr: 7.2900e-04 - 12s/epoch - 64ms/step
Epoch 5/20
194/194 - 14s - loss: 4.7732 - accuracy: 0.0667 - val_loss: 4.7422 - val_
accuracy: 0.0950 - lr: 6.5610e-04 - 14s/epoch - 70ms/step
Epoch 6/20
194/194 - 17s - loss: 4.7261 - accuracy: 0.0650 - val_loss: 4.7189 - val_
accuracy: 0.0900 - lr: 5.9049e-04 - 17s/epoch - 87ms/step
Epoch 7/20
194/194 - 21s - loss: 4.7162 - accuracy: 0.0680 - val_loss: 4.6998 - val_
accuracy: 0.0925 - lr: 5.3144e-04 - 21s/epoch - 108ms/step
Epoch 8/20
194/194 - 23s - loss: 4.6834 - accuracy: 0.0615 - val_loss: 4.6904 - val_
accuracy: 0.0925 - lr: 4.7830e-04 - 23s/epoch - 119ms/step
Epoch 9/20
194/194 - 18s - loss: 4.6914 - accuracy: 0.0608 - val_loss: 4.6843 - val_
accuracy: 0.0900 - lr: 4.3047e-04 - 18s/epoch - 93ms/step
Epoch 10/20
194/194 - 16s - loss: 4.6917 - accuracy: 0.0628 - val_loss: 4.6866 - val_
accuracy: 0.0925 - lr: 3.8742e-04 - 16s/epoch - 81ms/step
Epoch 11/20
194/194 - 15s - loss: 4.6697 - accuracy: 0.0644 - val_loss: 4.6773 - val_
accuracy: 0.0925 - lr: 3.4868e-04 - 15s/epoch - 75ms/step
Epoch 12/20
194/194 - 15s - loss: 4.6612 - accuracy: 0.0621 - val_loss: 4.6702 - val_
accuracy: 0.0900 - lr: 3.1381e-04 - 15s/epoch - 78ms/step
Epoch 13/20
194/194 - 13s - loss: 4.6661 - accuracy: 0.0673 - val_loss: 4.6635 - val_
accuracy: 0.0900 - lr: 2.8243e-04 - 13s/epoch - 67ms/step
Epoch 14/20
194/194 - 12s - loss: 4.6385 - accuracy: 0.0689 - val_loss: 4.6597 - val_
accuracy: 0.0925 - lr: 2.5419e-04 - 12s/epoch - 60ms/step
Epoch 15/20
194/194 - 12s - loss: 4.6548 - accuracy: 0.0680 - val_loss: 4.6613 - val_
```

```
accuracy: 0.0900 - lr: 2.2877e-04 - 12s/epoch - 63ms/step
Epoch 16/20
194/194 - 12s - loss: 4.6567 - accuracy: 0.0644 - val_loss: 4.6598 - val_
accuracy: 0.0925 - lr: 2.0589e-04 - 12s/epoch - 64ms/step
Epoch 17/20
194/194 - 13s - loss: 4.6520 - accuracy: 0.0654 - val_loss: 4.6595 - val_
accuracy: 0.0900 - lr: 1.8530e-04 - 13s/epoch - 65ms/step
Epoch 18/20
194/194 - 14s - loss: 4.6333 - accuracy: 0.0731 - val_loss: 4.6564 - val_
accuracy: 0.0900 - lr: 1.6677e-04 - 14s/epoch - 72ms/step
Epoch 19/20
194/194 - 18s - loss: 4.6382 - accuracy: 0.0676 - val_loss: 4.6532 - val_
accuracy: 0.0900 - lr: 1.5009e-04 - 18s/epoch - 92ms/step
Epoch 20/20
194/194 - 16s - loss: 4.6303 - accuracy: 0.0660 - val_loss: 4.6518 - val_
accuracy: 0.0900 - lr: 1.3509e-04 - 16s/epoch - 82ms/step
```

In [35]:

```
#Plotting model accuracy and loss
from matplotlib import pyplot as plt
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





In [38]:

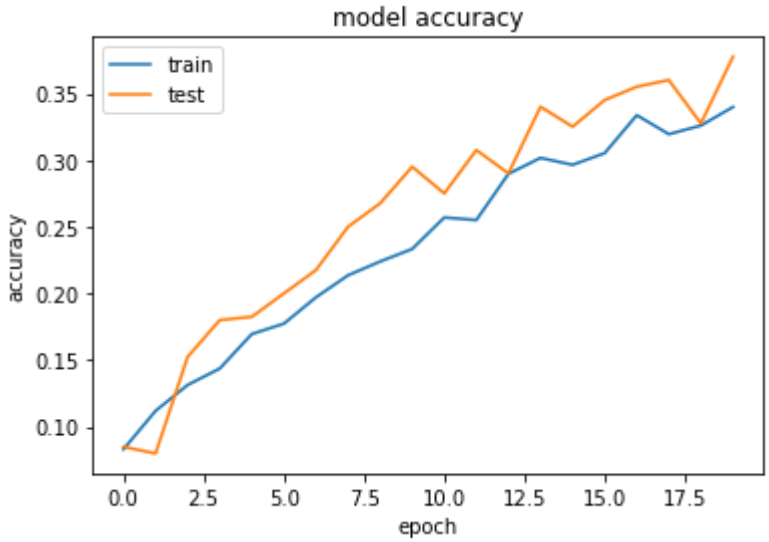
```
# The below code is adapted from https://www.kaggle.com/toregil/welcome-to-deep-learning-cn
# With batch normalization
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization
from tensorflow.keras.optimizers import Adam
from keras.callbacks import LearningRateScheduler

model = Sequential()

#Layer 1
model.add(BatchNormalization())
model.add(Conv2D(filters = 16, kernel_size = (3, 3), activation='relu',
                 input_shape = (80, 80, 1)))
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

#Layer 2
model.add(BatchNormalization())
model.add(Conv2D(filters = 32, kernel_size = (3, 3), activation='relu'))
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

#Layer 3
model.add(BatchNormalization())
model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation='relu'))
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

#Layer 4
model.add(BatchNormalization())
model.add(Conv2D(filters = 128, kernel_size = (3, 3), activation='relu'))
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(193, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer = Adam(learning_rate=1e-4), metric

annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)

hist = model.fit(datagen.flow(X_train, y_train, batch_size=16),
                              epochs=20, #Increase this when not on Kaggle kernel
                              verbose=2,  #1 for ETA, 0 for silent
                              validation_data=(X_test[:400,:], y_test[:400,:]), #For speed
                              callbacks=[annealer])
```

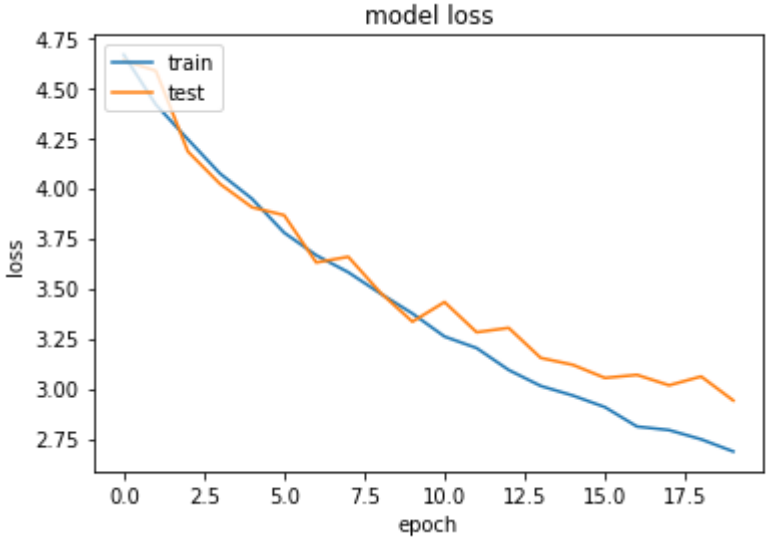
```
Epoch 1/20
194/194 - 11s - loss: 4.6644 - accuracy: 0.0832 - val_loss: 4.6358 - val_
accuracy: 0.0850 - lr: 0.0010 - 11s/epoch - 58ms/step
Epoch 2/20
194/194 - 12s - loss: 4.4160 - accuracy: 0.1120 - val_loss: 4.5845 - val_
accuracy: 0.0800 - lr: 9.0000e-04 - 12s/epoch - 64ms/step
```

```
Epoch 3/20
194/194 - 14s - loss: 4.2424 - accuracy: 0.1314 - val_loss: 4.1817 - val_
accuracy: 0.1525 - lr: 8.1000e-04 - 14s/epoch - 70ms/step
Epoch 4/20
194/194 - 15s - loss: 4.0724 - accuracy: 0.1437 - val_loss: 4.0205 - val_
accuracy: 0.1800 - lr: 7.2900e-04 - 15s/epoch - 77ms/step
Epoch 5/20
194/194 - 15s - loss: 3.9462 - accuracy: 0.1696 - val_loss: 3.9037 - val_
accuracy: 0.1825 - lr: 6.5610e-04 - 15s/epoch - 80ms/step
Epoch 6/20
194/194 - 17s - loss: 3.7772 - accuracy: 0.1773 - val_loss: 3.8644 - val_
accuracy: 0.2000 - lr: 5.9049e-04 - 17s/epoch - 90ms/step
Epoch 7/20
194/194 - 23s - loss: 3.6647 - accuracy: 0.1971 - val_loss: 3.6288 - val_
accuracy: 0.2175 - lr: 5.3144e-04 - 23s/epoch - 119ms/step
Epoch 8/20
194/194 - 26s - loss: 3.5799 - accuracy: 0.2136 - val_loss: 3.6571 - val_
accuracy: 0.2500 - lr: 4.7830e-04 - 26s/epoch - 133ms/step
Epoch 9/20
194/194 - 22s - loss: 3.4741 - accuracy: 0.2239 - val_loss: 3.4782 - val_
accuracy: 0.2675 - lr: 4.3047e-04 - 22s/epoch - 115ms/step
Epoch 10/20
194/194 - 18s - loss: 3.3744 - accuracy: 0.2333 - val_loss: 3.3337 - val_
accuracy: 0.2950 - lr: 3.8742e-04 - 18s/epoch - 95ms/step
Epoch 11/20
194/194 - 15s - loss: 3.2596 - accuracy: 0.2570 - val_loss: 3.4315 - val_
accuracy: 0.2750 - lr: 3.4868e-04 - 15s/epoch - 79ms/step
Epoch 12/20
194/194 - 15s - loss: 3.2022 - accuracy: 0.2550 - val_loss: 3.2810 - val_
accuracy: 0.3075 - lr: 3.1381e-04 - 15s/epoch - 77ms/step
Epoch 13/20
194/194 - 13s - loss: 3.0929 - accuracy: 0.2896 - val_loss: 3.3027 - val_
accuracy: 0.2900 - lr: 2.8243e-04 - 13s/epoch - 69ms/step
Epoch 14/20
194/194 - 13s - loss: 3.0132 - accuracy: 0.3016 - val_loss: 3.1530 - val_
accuracy: 0.3400 - lr: 2.5419e-04 - 13s/epoch - 66ms/step
Epoch 15/20
194/194 - 14s - loss: 2.9655 - accuracy: 0.2964 - val_loss: 3.1186 - val_
accuracy: 0.3250 - lr: 2.2877e-04 - 14s/epoch - 70ms/step
Epoch 16/20
194/194 - 14s - loss: 2.9077 - accuracy: 0.3052 - val_loss: 3.0533 - val_
accuracy: 0.3450 - lr: 2.0589e-04 - 14s/epoch - 73ms/step
Epoch 17/20
194/194 - 15s - loss: 2.8108 - accuracy: 0.3337 - val_loss: 3.0685 - val_
accuracy: 0.3550 - lr: 1.8530e-04 - 15s/epoch - 78ms/step
Epoch 18/20
194/194 - 17s - loss: 2.7933 - accuracy: 0.3194 - val_loss: 3.0165 - val_
accuracy: 0.3600 - lr: 1.6677e-04 - 17s/epoch - 89ms/step
Epoch 19/20
194/194 - 18s - loss: 2.7472 - accuracy: 0.3259 - val_loss: 3.0605 - val_
accuracy: 0.3275 - lr: 1.5009e-04 - 18s/epoch - 95ms/step
Epoch 20/20
194/194 - 16s - loss: 2.6866 - accuracy: 0.3398 - val_loss: 2.9409 - val_
accuracy: 0.3775 - lr: 1.3509e-04 - 16s/epoch - 82ms/step
```

In [39]:

```
#Plotting model accuracy and loss
from matplotlib import pyplot as plt
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





In [41]:

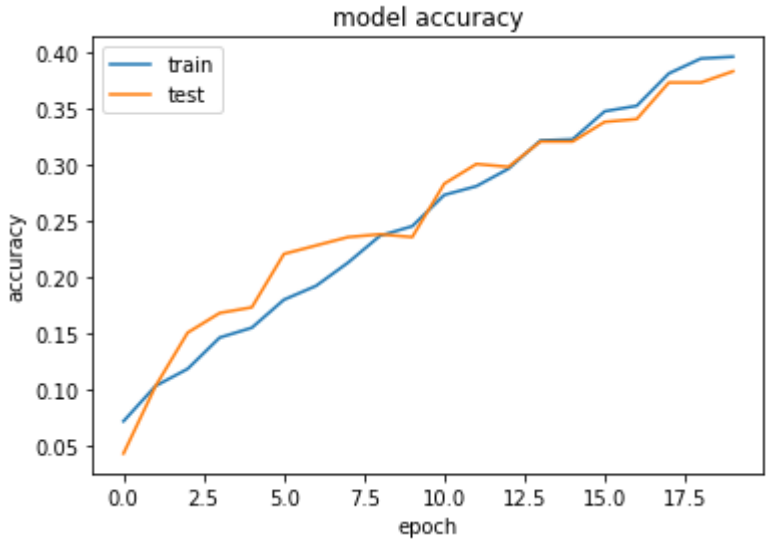
```
# The below code is adapted from https://www.kaggle.com/toregil/welcome-to-deep-learning-cn
# with batch normalization
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization
from tensorflow.keras.optimizers import Adam
from keras.callbacks import LearningRateScheduler

model = Sequential()

#Layer 1
model.add(BatchNormalization())
model.add(Conv2D(filters = 16, kernel_size = (3, 3), activation='relu',
                 input_shape = (80, 80, 1)))
model.add(BatchNormalization())
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

#Layer 2
model.add(BatchNormalization())
model.add(Conv2D(filters = 32, kernel_size = (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(193, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer = Adam(learning_rate=1e-4), metric

annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)

hist = model.fit(datagen.flow(X_train, y_train, batch_size=16),
                              epochs=20, #Increase this when not on Kaggle kernel
                              verbose=2,  #1 for ETA, 0 for silent
                              validation_data=(X_test[:400,:], y_test[:400,:]), #For speed
                              callbacks=[annealer])
```

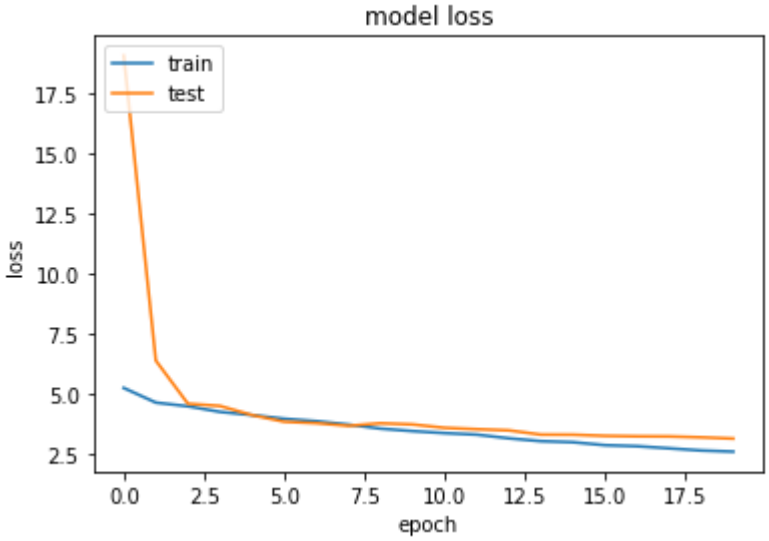
```
Epoch 1/20
194/194 - 20s - loss: 5.2489 - accuracy: 0.0712 - val_loss: 19.1072 - val_ac
curacy: 0.0425 - lr: 0.0010 - 20s/epoch - 102ms/step
Epoch 2/20
194/194 - 19s - loss: 4.6380 - accuracy: 0.1026 - val_loss: 6.3898 - val_acc
uracy: 0.1025 - lr: 9.0000e-04 - 19s/epoch - 96ms/step
Epoch 3/20
194/194 - 18s - loss: 4.4849 - accuracy: 0.1178 - val_loss: 4.5907 - val_acc
uracy: 0.1500 - lr: 8.1000e-04 - 18s/epoch - 91ms/step
Epoch 4/20
194/194 - 19s - loss: 4.2513 - accuracy: 0.1456 - val_loss: 4.4985 - val_acc
uracy: 0.1675 - lr: 7.2900e-04 - 19s/epoch - 100ms/step
Epoch 5/20
194/194 - 21s - loss: 4.1172 - accuracy: 0.1544 - val_loss: 4.1166 - val_acc
uracy: 0.1725 - lr: 6.5610e-04 - 21s/epoch - 110ms/step
Epoch 6/20
194/194 - 23s - loss: 3.9637 - accuracy: 0.1793 - val_loss: 3.8408 - val_acc
```

```
uracy: 0.2200 - lr: 5.9049e-04 - 23s/epoch - 119ms/step
Epoch 7/20
194/194 - 30s - loss: 3.8637 - accuracy: 0.1916 - val_loss: 3.7818 - val_acc
uracy: 0.2275 - lr: 5.3144e-04 - 30s/epoch - 154ms/step
Epoch 8/20
194/194 - 23s - loss: 3.7218 - accuracy: 0.2123 - val_loss: 3.6700 - val_acc
uracy: 0.2350 - lr: 4.7830e-04 - 23s/epoch - 118ms/step
Epoch 9/20
194/194 - 20s - loss: 3.5532 - accuracy: 0.2362 - val_loss: 3.7750 - val_acc
uracy: 0.2375 - lr: 4.3047e-04 - 20s/epoch - 101ms/step
Epoch 10/20
194/194 - 20s - loss: 3.4493 - accuracy: 0.2447 - val_loss: 3.7281 - val_acc
uracy: 0.2350 - lr: 3.8742e-04 - 20s/epoch - 101ms/step
Epoch 11/20
194/194 - 20s - loss: 3.3657 - accuracy: 0.2725 - val_loss: 3.5862 - val_acc
uracy: 0.2825 - lr: 3.4868e-04 - 20s/epoch - 104ms/step
Epoch 12/20
194/194 - 21s - loss: 3.3025 - accuracy: 0.2803 - val_loss: 3.5289 - val_acc
uracy: 0.3000 - lr: 3.1381e-04 - 21s/epoch - 110ms/step
Epoch 13/20
194/194 - 27s - loss: 3.1491 - accuracy: 0.2958 - val_loss: 3.4802 - val_acc
uracy: 0.2975 - lr: 2.8243e-04 - 27s/epoch - 137ms/step
Epoch 14/20
194/194 - 23s - loss: 3.0297 - accuracy: 0.3210 - val_loss: 3.3037 - val_acc
uracy: 0.3200 - lr: 2.5419e-04 - 23s/epoch - 117ms/step
Epoch 15/20
194/194 - 21s - loss: 2.9871 - accuracy: 0.3220 - val_loss: 3.2993 - val_acc
uracy: 0.3200 - lr: 2.2877e-04 - 21s/epoch - 108ms/step
Epoch 16/20
194/194 - 20s - loss: 2.8625 - accuracy: 0.3469 - val_loss: 3.2516 - val_acc
uracy: 0.3375 - lr: 2.0589e-04 - 20s/epoch - 103ms/step
Epoch 17/20
194/194 - 18s - loss: 2.8200 - accuracy: 0.3518 - val_loss: 3.2326 - val_acc
uracy: 0.3400 - lr: 1.8530e-04 - 18s/epoch - 95ms/step
Epoch 18/20
194/194 - 18s - loss: 2.7311 - accuracy: 0.3806 - val_loss: 3.2263 - val_acc
uracy: 0.3725 - lr: 1.6677e-04 - 18s/epoch - 94ms/step
Epoch 19/20
194/194 - 19s - loss: 2.6407 - accuracy: 0.3939 - val_loss: 3.1885 - val_acc
uracy: 0.3725 - lr: 1.5009e-04 - 19s/epoch - 100ms/step
Epoch 20/20
194/194 - 22s - loss: 2.5931 - accuracy: 0.3955 - val_loss: 3.1377 - val_acc
uracy: 0.3825 - lr: 1.3509e-04 - 22s/epoch - 113ms/step
```

In [42]:

```
#Plotting model accuracy and loss
from matplotlib import pyplot as plt
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





In [46]:

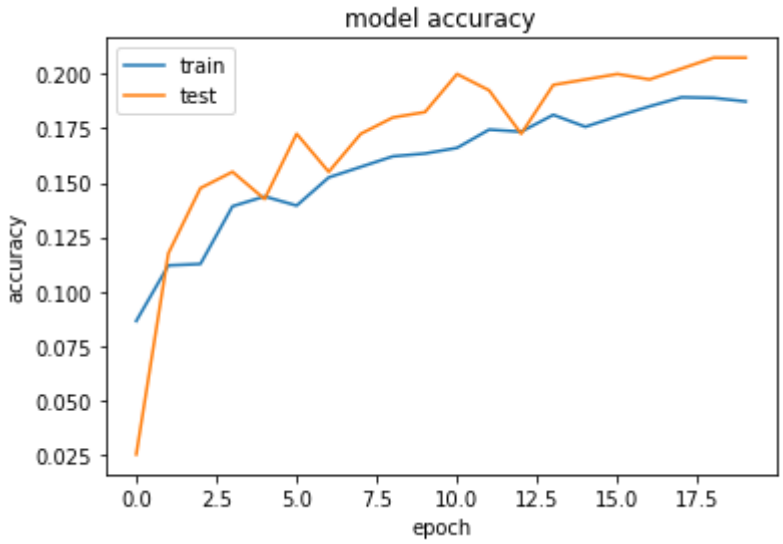
```
# The below code is adapted from https://www.kaggle.com/toregil/welcome-to-deep-learning-cn
# RMSprop
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization
from tensorflow.keras.optimizers import RMSprop
from keras.callbacks import LearningRateScheduler

model = Sequential()

#Layer 1
model.add(BatchNormalization())
model.add(Conv2D(filters = 16, kernel_size = (3, 3), activation='relu',
                 input_shape = (80, 80, 1)))
model.add(BatchNormalization())
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

#Layer 2
model.add(BatchNormalization())
model.add(Conv2D(filters = 32, kernel_size = (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(193, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer = RMSprop(learning_rate=1e-4), met

annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)

hist = model.fit(datagen.flow(X_train, y_train, batch_size=16),
                              epochs=20, #Increase this when not on Kaggle kernel
                              verbose=2,  #1 for ETA, 0 for silent
                              validation_data=(X_test[:400,:], y_test[:400,:]), #For speed
                              callbacks=[annealer])
```

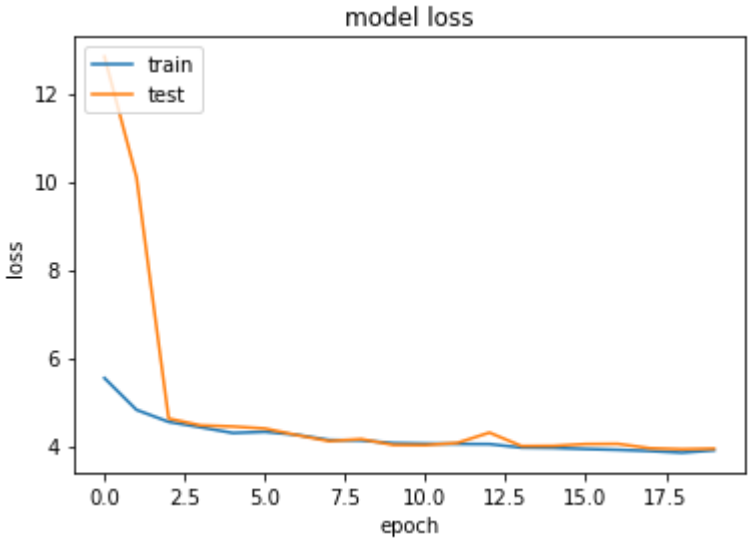
```
Epoch 1/20
194/194 - 24s - loss: 5.5376 - accuracy: 0.0864 - val_loss: 12.8307 - val_ac
curacy: 0.0250 - lr: 0.0010 - 24s/epoch - 121ms/step
Epoch 2/20
194/194 - 22s - loss: 4.8172 - accuracy: 0.1120 - val_loss: 10.0872 - val_ac
curacy: 0.1175 - lr: 9.0000e-04 - 22s/epoch - 112ms/step
Epoch 3/20
194/194 - 24s - loss: 4.5451 - accuracy: 0.1126 - val_loss: 4.6204 - val_acc
uracy: 0.1475 - lr: 8.1000e-04 - 24s/epoch - 122ms/step
Epoch 4/20
194/194 - 24s - loss: 4.4213 - accuracy: 0.1392 - val_loss: 4.4648 - val_acc
uracy: 0.1550 - lr: 7.2900e-04 - 24s/epoch - 124ms/step
Epoch 5/20
194/194 - 27s - loss: 4.2933 - accuracy: 0.1437 - val_loss: 4.4397 - val_acc
uracy: 0.1425 - lr: 6.5610e-04 - 27s/epoch - 139ms/step
Epoch 6/20
```

```
194/194 - 31s - loss: 4.3191 - accuracy: 0.1395 - val_loss: 4.3945 - val_acc
uracy: 0.1725 - lr: 5.9049e-04 - 31s/epoch - 160ms/step
Epoch 7/20
194/194 - 25s - loss: 4.2507 - accuracy: 0.1524 - val_loss: 4.2445 - val_acc
uracy: 0.1550 - lr: 5.3144e-04 - 25s/epoch - 127ms/step
Epoch 8/20
194/194 - 23s - loss: 4.1344 - accuracy: 0.1573 - val_loss: 4.1082 - val_acc
uracy: 0.1725 - lr: 4.7830e-04 - 23s/epoch - 117ms/step
Epoch 9/20
194/194 - 22s - loss: 4.1233 - accuracy: 0.1621 - val_loss: 4.1545 - val_acc
uracy: 0.1800 - lr: 4.3047e-04 - 22s/epoch - 112ms/step
Epoch 10/20
194/194 - 21s - loss: 4.0632 - accuracy: 0.1634 - val_loss: 4.0213 - val_acc
uracy: 0.1825 - lr: 3.8742e-04 - 21s/epoch - 111ms/step
Epoch 11/20
194/194 - 22s - loss: 4.0519 - accuracy: 0.1660 - val_loss: 4.0174 - val_acc
uracy: 0.2000 - lr: 3.4868e-04 - 22s/epoch - 116ms/step
Epoch 12/20
194/194 - 24s - loss: 4.0438 - accuracy: 0.1744 - val_loss: 4.0679 - val_acc
uracy: 0.1925 - lr: 3.1381e-04 - 24s/epoch - 123ms/step
Epoch 13/20
194/194 - 31s - loss: 4.0408 - accuracy: 0.1735 - val_loss: 4.3014 - val_acc
uracy: 0.1725 - lr: 2.8243e-04 - 31s/epoch - 160ms/step
Epoch 14/20
194/194 - 28s - loss: 3.9612 - accuracy: 0.1812 - val_loss: 3.9964 - val_acc
uracy: 0.1950 - lr: 2.5419e-04 - 28s/epoch - 145ms/step
Epoch 15/20
194/194 - 24s - loss: 3.9535 - accuracy: 0.1757 - val_loss: 4.0000 - val_acc
uracy: 0.1975 - lr: 2.2877e-04 - 24s/epoch - 124ms/step
Epoch 16/20
194/194 - 24s - loss: 3.9299 - accuracy: 0.1806 - val_loss: 4.0415 - val_acc
uracy: 0.2000 - lr: 2.0589e-04 - 24s/epoch - 123ms/step
Epoch 17/20
194/194 - 24s - loss: 3.9109 - accuracy: 0.1851 - val_loss: 4.0462 - val_acc
uracy: 0.1975 - lr: 1.8530e-04 - 24s/epoch - 126ms/step
Epoch 18/20
194/194 - 28s - loss: 3.8841 - accuracy: 0.1893 - val_loss: 3.9465 - val_acc
uracy: 0.2025 - lr: 1.6677e-04 - 28s/epoch - 145ms/step
Epoch 19/20
194/194 - 29s - loss: 3.8470 - accuracy: 0.1890 - val_loss: 3.9251 - val_acc
uracy: 0.2075 - lr: 1.5009e-04 - 29s/epoch - 150ms/step
Epoch 20/20
194/194 - 27s - loss: 3.8999 - accuracy: 0.1874 - val_loss: 3.9378 - val_acc
uracy: 0.2075 - lr: 1.3509e-04 - 27s/epoch - 138ms/step
```

In [47]:

```
#Plotting model accuracy and loss
from matplotlib import pyplot as plt
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





In [48]:

```
# The below code is adapted from https://www.kaggle.com/toregil/welcome-to-deep-learning-cn
# Changing the learning rate for Adam optimizer
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization
from tensorflow.keras.optimizers import Adam
from keras.callbacks import LearningRateScheduler

model = Sequential()

#Layer 1
model.add(Conv2D(filters = 16, kernel_size = (3, 3), activation='relu',
                 input_shape = (80, 80, 1)))
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

#Layer 2
model.add(Conv2D(filters = 32, kernel_size = (3, 3), activation='relu'))
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(193, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer = Adam(learning_rate=3e-4), metric

annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)

model.summary()

hist = model.fit(datagen.flow(X_train, y_train, batch_size=16),
                              epochs=20, #Increase this when not on Kaggle kernel
                              verbose=2,  #1 for ETA, 0 for silent
                              validation_data=(X_test[:400,:], y_test[:400,:]), #For speed
                              callbacks=[annealer])
```

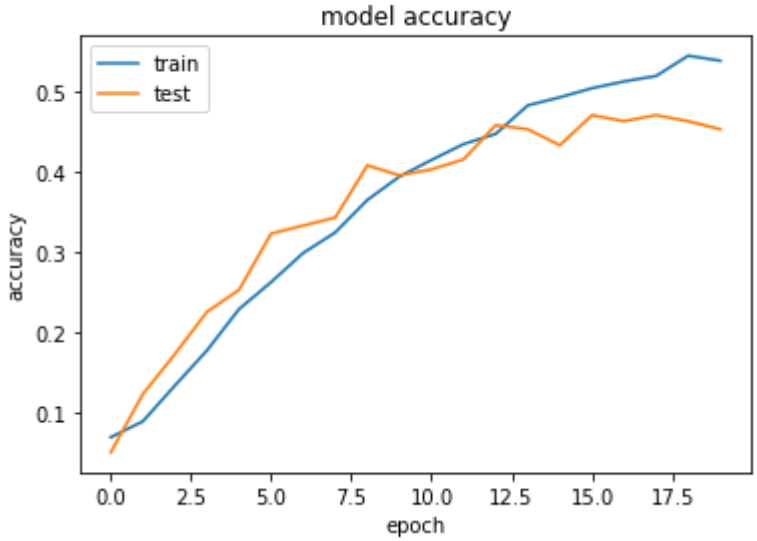
```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_40 (Conv2D)          (None, 78, 78, 16)        160

 max_pooling2d_40 (MaxPoolin  (None, 39, 39, 16)       0
 g2D)

 dropout_72 (Dropout)        (None, 39, 39, 16)        0

 conv2d_41 (Conv2D)          (None, 37, 37, 32)        4640

 max_pooling2d_41 (MaxPoolin  (None, 18, 18, 32)       0
 g2D)

 dropout_73 (Dropout)        (None, 18, 18, 32)        0
```

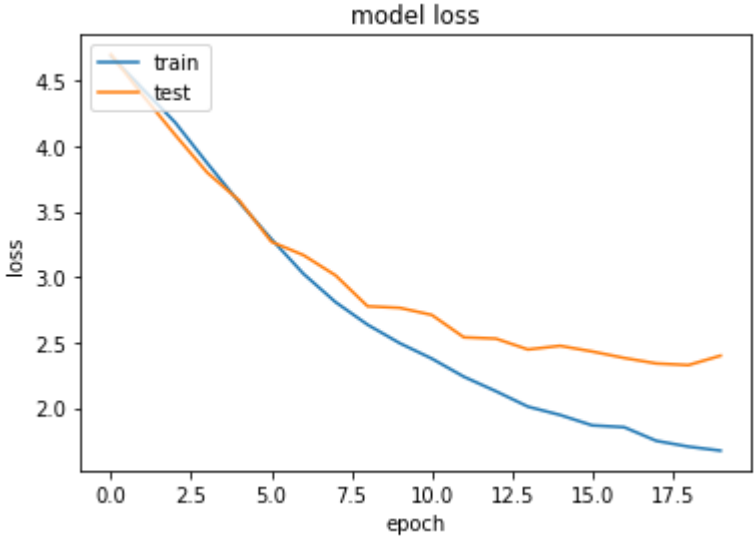
```
 flatten_16 (Flatten)        (None, 10368)             0

 dense_48 (Dense)            (None, 512)               5308928

 dropout_74 (Dropout)        (None, 512)               0

 dense_49 (Dense)            (None, 1024)              525312

 dropout_75 (Dropout)        (None, 1024)              0

 dense_50 (Dense)            (None, 193)               197825

=================================================================
Total params: 6,036,865
Trainable params: 6,036,865
Non-trainable params: 0

_________________________________________________________________
Epoch 1/20
194/194 - 13s - loss: 4.6881 - accuracy: 0.0689 - val_loss: 4.6987 - val_acc
uracy: 0.0500 - lr: 0.0010 - 13s/epoch - 69ms/step
Epoch 2/20
194/194 - 13s - loss: 4.4341 - accuracy: 0.0887 - val_loss: 4.3853 - val_acc
uracy: 0.1225 - lr: 9.0000e-04 - 13s/epoch - 66ms/step
Epoch 3/20
194/194 - 13s - loss: 4.1807 - accuracy: 0.1333 - val_loss: 4.0856 - val_acc
uracy: 0.1725 - lr: 8.1000e-04 - 13s/epoch - 67ms/step
Epoch 4/20
194/194 - 13s - loss: 3.8695 - accuracy: 0.1770 - val_loss: 3.7977 - val_acc
uracy: 0.2250 - lr: 7.2900e-04 - 13s/epoch - 68ms/step
Epoch 5/20
194/194 - 13s - loss: 3.5702 - accuracy: 0.2285 - val_loss: 3.5840 - val_acc
uracy: 0.2525 - lr: 6.5610e-04 - 13s/epoch - 68ms/step
Epoch 6/20
194/194 - 13s - loss: 3.2902 - accuracy: 0.2621 - val_loss: 3.2692 - val_acc
uracy: 0.3225 - lr: 5.9049e-04 - 13s/epoch - 68ms/step
Epoch 7/20
194/194 - 13s - loss: 3.0245 - accuracy: 0.2984 - val_loss: 3.1671 - val_acc
uracy: 0.3325 - lr: 5.3144e-04 - 13s/epoch - 69ms/step
Epoch 8/20
194/194 - 14s - loss: 2.8083 - accuracy: 0.3239 - val_loss: 3.0126 - val_acc
uracy: 0.3425 - lr: 4.7830e-04 - 14s/epoch - 73ms/step
Epoch 9/20
194/194 - 15s - loss: 2.6356 - accuracy: 0.3644 - val_loss: 2.7763 - val_acc
uracy: 0.4075 - lr: 4.3047e-04 - 15s/epoch - 77ms/step
Epoch 10/20
194/194 - 16s - loss: 2.4947 - accuracy: 0.3935 - val_loss: 2.7645 - val_acc
uracy: 0.3950 - lr: 3.8742e-04 - 16s/epoch - 84ms/step
Epoch 11/20
194/194 - 20s - loss: 2.3769 - accuracy: 0.4142 - val_loss: 2.7104 - val_acc
uracy: 0.4025 - lr: 3.4868e-04 - 20s/epoch - 101ms/step
Epoch 12/20
194/194 - 18s - loss: 2.2384 - accuracy: 0.4340 - val_loss: 2.5393 - val_acc
uracy: 0.4150 - lr: 3.1381e-04 - 18s/epoch - 93ms/step
Epoch 13/20
194/194 - 15s - loss: 2.1275 - accuracy: 0.4466 - val_loss: 2.5293 - val_acc
uracy: 0.4575 - lr: 2.8243e-04 - 15s/epoch - 78ms/step
Epoch 14/20
194/194 - 14s - loss: 2.0096 - accuracy: 0.4822 - val_loss: 2.4464 - val_acc
uracy: 0.4525 - lr: 2.5419e-04 - 14s/epoch - 73ms/step
Epoch 15/20
194/194 - 14s - loss: 1.9463 - accuracy: 0.4922 - val_loss: 2.4744 - val_acc
```

```
uracy: 0.4325 - lr: 2.2877e-04 - 14s/epoch - 71ms/step
Epoch 16/20
194/194 - 14s - loss: 1.8670 - accuracy: 0.5036 - val_loss: 2.4311 - val_acc
uracy: 0.4700 - lr: 2.0589e-04 - 14s/epoch - 71ms/step
Epoch 17/20
194/194 - 15s - loss: 1.8527 - accuracy: 0.5120 - val_loss: 2.3815 - val_acc
uracy: 0.4625 - lr: 1.8530e-04 - 15s/epoch - 77ms/step
Epoch 18/20
194/194 - 15s - loss: 1.7486 - accuracy: 0.5188 - val_loss: 2.3395 - val_acc
uracy: 0.4700 - lr: 1.6677e-04 - 15s/epoch - 78ms/step
Epoch 19/20
194/194 - 16s - loss: 1.7045 - accuracy: 0.5440 - val_loss: 2.3271 - val_acc
uracy: 0.4625 - lr: 1.5009e-04 - 16s/epoch - 84ms/step
Epoch 20/20
194/194 - 20s - loss: 1.6737 - accuracy: 0.5379 - val_loss: 2.3988 - val_acc
uracy: 0.4525 - lr: 1.3509e-04 - 20s/epoch - 103ms/step
```

In [49]:

```
#Plotting model accuracy and loss
from matplotlib import pyplot as plt
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





In [53]:

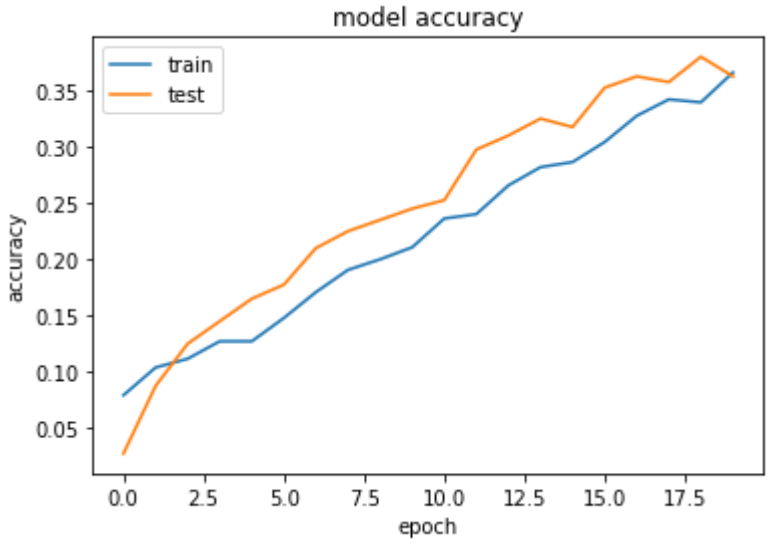
```
# The below code is adapted from https://www.kaggle.com/toregil/welcome-to-deep-learning-cn
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization
from tensorflow.keras.optimizers import Adam
from keras.callbacks import LearningRateScheduler

model = Sequential()

#Layer 1
model.add(BatchNormalization())
model.add(Conv2D(filters = 16, kernel_size = (3, 3), activation='relu',
                 input_shape = (80, 80, 1)))
model.add(BatchNormalization())
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

#Layer 2
model.add(BatchNormalization())
model.add(Conv2D(filters = 32, kernel_size = (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(193, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer = Adam(learning_rate=3e-4), metric

annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)

hist = model.fit(datagen.flow(X_train, y_train, batch_size=16),
                           epochs=20, #Increase this when not on Kaggle kernel
                           verbose=2,  #1 for ETA, 0 for silent
                           validation_data=(X_test[:400,:], y_test[:400,:]), #For speed
                           callbacks=[annealer])
```

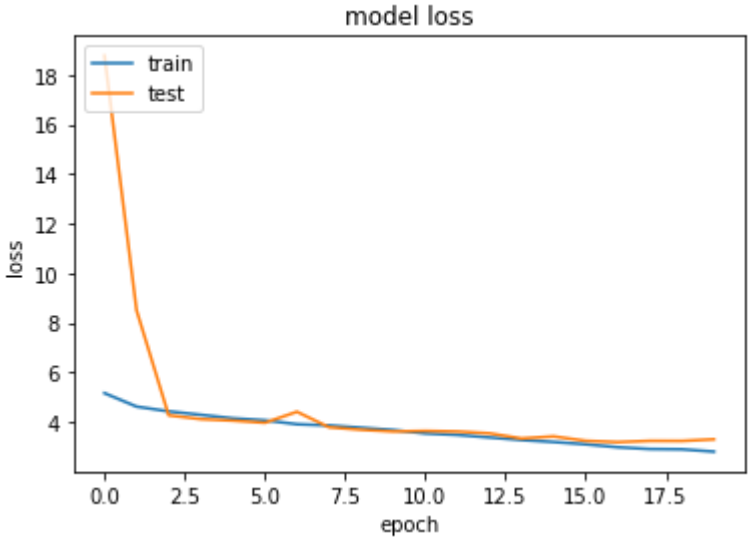
```
Epoch 1/20
194/194 - 18s - loss: 5.1638 - accuracy: 0.0793 - val_loss: 18.7746 - val
_accuracy: 0.0275 - lr: 0.0010 - 18s/epoch - 95ms/step
Epoch 2/20
194/194 - 18s - loss: 4.6209 - accuracy: 0.1039 - val_loss: 8.5049 - val_
accuracy: 0.0875 - lr: 9.0000e-04 - 18s/epoch - 91ms/step
Epoch 3/20
194/194 - 18s - loss: 4.4283 - accuracy: 0.1117 - val_loss: 4.2706 - val_
accuracy: 0.1250 - lr: 8.1000e-04 - 18s/epoch - 91ms/step
Epoch 4/20
194/194 - 18s - loss: 4.2896 - accuracy: 0.1272 - val_loss: 4.1161 - val_
accuracy: 0.1450 - lr: 7.2900e-04 - 18s/epoch - 91ms/step
Epoch 5/20
194/194 - 19s - loss: 4.1481 - accuracy: 0.1272 - val_loss: 4.0568 - val_
accuracy: 0.1650 - lr: 6.5610e-04 - 19s/epoch - 96ms/step
Epoch 6/20
194/194 - 20s - loss: 4.0675 - accuracy: 0.1479 - val_loss: 3.9699 - val_
```

```
accuracy: 0.1775 - lr: 5.9049e-04 - 20s/epoch - 104ms/step
Epoch 7/20
194/194 - 24s - loss: 3.9103 - accuracy: 0.1709 - val_loss: 4.4121 - val_
accuracy: 0.2100 - lr: 5.3144e-04 - 24s/epoch - 125ms/step
Epoch 8/20
194/194 - 37s - loss: 3.8598 - accuracy: 0.1906 - val_loss: 3.7868 - val_
accuracy: 0.2250 - lr: 4.7830e-04 - 37s/epoch - 189ms/step
Epoch 9/20
194/194 - 21s - loss: 3.7689 - accuracy: 0.2000 - val_loss: 3.6875 - val_
accuracy: 0.2350 - lr: 4.3047e-04 - 21s/epoch - 108ms/step
Epoch 10/20
194/194 - 19s - loss: 3.6760 - accuracy: 0.2107 - val_loss: 3.6178 - val_
accuracy: 0.2450 - lr: 3.8742e-04 - 19s/epoch - 99ms/step
Epoch 11/20
194/194 - 19s - loss: 3.5388 - accuracy: 0.2362 - val_loss: 3.6392 - val_
accuracy: 0.2525 - lr: 3.4868e-04 - 19s/epoch - 96ms/step
Epoch 12/20
194/194 - 18s - loss: 3.4794 - accuracy: 0.2401 - val_loss: 3.6151 - val_
accuracy: 0.2975 - lr: 3.1381e-04 - 18s/epoch - 92ms/step
Epoch 13/20
194/194 - 18s - loss: 3.3771 - accuracy: 0.2657 - val_loss: 3.5410 - val_
accuracy: 0.3100 - lr: 2.8243e-04 - 18s/epoch - 95ms/step
Epoch 14/20
194/194 - 20s - loss: 3.2741 - accuracy: 0.2819 - val_loss: 3.3355 - val_
accuracy: 0.3250 - lr: 2.5419e-04 - 20s/epoch - 102ms/step
Epoch 15/20
194/194 - 21s - loss: 3.1942 - accuracy: 0.2864 - val_loss: 3.4188 - val_
accuracy: 0.3175 - lr: 2.2877e-04 - 21s/epoch - 109ms/step
Epoch 16/20
194/194 - 23s - loss: 3.1009 - accuracy: 0.3042 - val_loss: 3.2393 - val_
accuracy: 0.3525 - lr: 2.0589e-04 - 23s/epoch - 118ms/step
Epoch 17/20
194/194 - 20s - loss: 2.9822 - accuracy: 0.3275 - val_loss: 3.1888 - val_
accuracy: 0.3625 - lr: 1.8530e-04 - 20s/epoch - 102ms/step
Epoch 18/20
194/194 - 18s - loss: 2.9101 - accuracy: 0.3421 - val_loss: 3.2382 - val_
accuracy: 0.3575 - lr: 1.6677e-04 - 18s/epoch - 95ms/step
Epoch 19/20
194/194 - 18s - loss: 2.8980 - accuracy: 0.3395 - val_loss: 3.2361 - val_
accuracy: 0.3800 - lr: 1.5009e-04 - 18s/epoch - 92ms/step
Epoch 20/20
194/194 - 18s - loss: 2.8045 - accuracy: 0.3660 - val_loss: 3.3014 - val_
accuracy: 0.3625 - lr: 1.3509e-04 - 18s/epoch - 94ms/step
```

In [54]:

```
#Plotting model accuracy and loss
from matplotlib import pyplot as plt
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





In [55]:

```
# The below code is adapted from https://www.kaggle.com/toregil/welcome-to-deep-learning-cn
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization
from tensorflow.keras.optimizers import Adam
from keras.callbacks import LearningRateScheduler

model = Sequential()

#Layer 1
model.add(Conv2D(filters = 16, kernel_size = (3, 3), activation='relu',
                 input_shape = (80, 80, 1)))
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

#Layer 2
model.add(Conv2D(filters = 32, kernel_size = (3, 3), activation='relu'))
model.add(MaxPool2D(strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(193, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer = Adam(learning_rate=4e-4), metric

annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)

model.summary()

hist = model.fit(datagen.flow(X_train, y_train, batch_size=16),
                           epochs=20, #Increase this when not on Kaggle kernel
                           verbose=2,  #1 for ETA, 0 for silent
                           validation_data=(X_test[:400,:], y_test[:400,:]), #For speed
                           callbacks=[annealer])
```

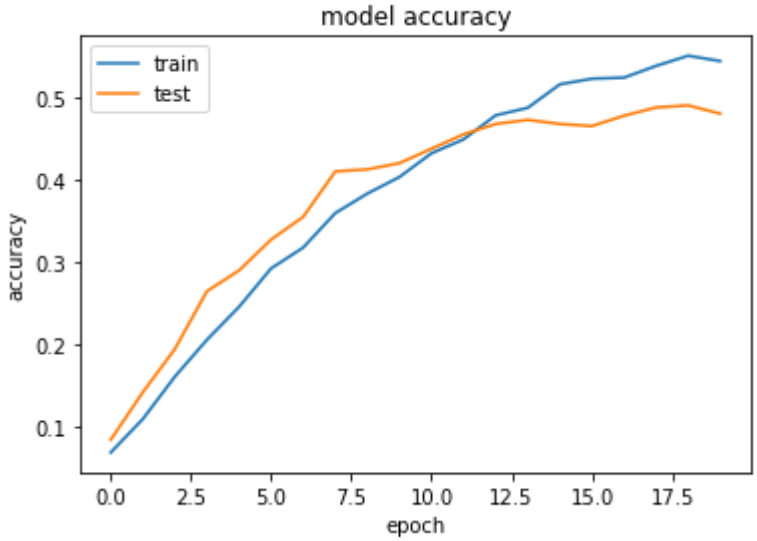
```
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_49 (Conv2D)          (None, 78, 78, 16)        160

 max_pooling2d_49 (MaxPoolin  (None, 39, 39, 16)       0
 g2D)

 dropout_89 (Dropout)        (None, 39, 39, 16)        0

 conv2d_50 (Conv2D)          (None, 37, 37, 32)        4640

 max_pooling2d_50 (MaxPoolin  (None, 18, 18, 32)       0
 g2D)

 dropout_90 (Dropout)        (None, 18, 18, 32)        0

 flatten_20 (Flatten)        (None, 10368)             0
```

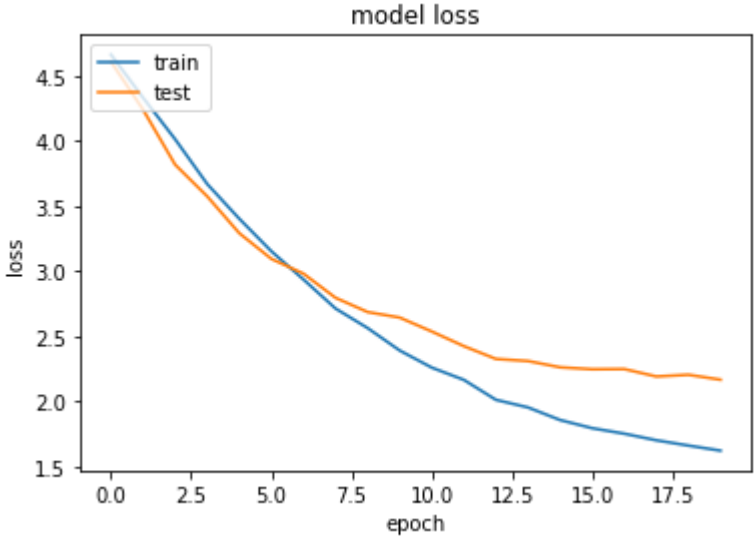
```
 dense_60 (Dense)            (None, 512)               5308928

 dropout_91 (Dropout)        (None, 512)               0

 dense_61 (Dense)            (None, 1024)              525312

 dropout_92 (Dropout)        (None, 1024)              0

 dense_62 (Dense)            (None, 193)               197825

=================================================================
Total params: 6,036,865
Trainable params: 6,036,865
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
194/194 - 13s - loss: 4.6611 - accuracy: 0.0696 - val_loss: 4.6068 - val_acc
uracy: 0.0850 - lr: 0.0010 - 13s/epoch - 69ms/step
Epoch 2/20
194/194 - 13s - loss: 4.3303 - accuracy: 0.1100 - val_loss: 4.2384 - val_acc
uracy: 0.1425 - lr: 9.0000e-04 - 13s/epoch - 66ms/step
Epoch 3/20
194/194 - 13s - loss: 4.0116 - accuracy: 0.1615 - val_loss: 3.8166 - val_acc
uracy: 0.1950 - lr: 8.1000e-04 - 13s/epoch - 67ms/step
Epoch 4/20
194/194 - 13s - loss: 3.6680 - accuracy: 0.2061 - val_loss: 3.5733 - val_acc
uracy: 0.2650 - lr: 7.2900e-04 - 13s/epoch - 67ms/step
Epoch 5/20
194/194 - 13s - loss: 3.4016 - accuracy: 0.2460 - val_loss: 3.2906 - val_acc
uracy: 0.2900 - lr: 6.5610e-04 - 13s/epoch - 69ms/step
Epoch 6/20
194/194 - 14s - loss: 3.1520 - accuracy: 0.2926 - val_loss: 3.0937 - val_acc
uracy: 0.3275 - lr: 5.9049e-04 - 14s/epoch - 72ms/step
Epoch 7/20
194/194 - 15s - loss: 2.9356 - accuracy: 0.3178 - val_loss: 2.9804 - val_acc
uracy: 0.3550 - lr: 5.3144e-04 - 15s/epoch - 77ms/step
Epoch 8/20
194/194 - 16s - loss: 2.7127 - accuracy: 0.3595 - val_loss: 2.7952 - val_acc
uracy: 0.4100 - lr: 4.7830e-04 - 16s/epoch - 83ms/step
Epoch 9/20
194/194 - 17s - loss: 2.5630 - accuracy: 0.3832 - val_loss: 2.6859 - val_acc
uracy: 0.4125 - lr: 4.3047e-04 - 17s/epoch - 88ms/step
Epoch 10/20
194/194 - 16s - loss: 2.3907 - accuracy: 0.4032 - val_loss: 2.6434 - val_acc
uracy: 0.4200 - lr: 3.8742e-04 - 16s/epoch - 80ms/step
Epoch 11/20
194/194 - 16s - loss: 2.2595 - accuracy: 0.4320 - val_loss: 2.5365 - val_acc
uracy: 0.4375 - lr: 3.4868e-04 - 16s/epoch - 80ms/step
Epoch 12/20
194/194 - 14s - loss: 2.1655 - accuracy: 0.4489 - val_loss: 2.4241 - val_acc
uracy: 0.4550 - lr: 3.1381e-04 - 14s/epoch - 70ms/step
Epoch 13/20
194/194 - 13s - loss: 2.0123 - accuracy: 0.4780 - val_loss: 2.3260 - val_acc
uracy: 0.4675 - lr: 2.8243e-04 - 13s/epoch - 69ms/step
Epoch 14/20
194/194 - 13s - loss: 1.9539 - accuracy: 0.4871 - val_loss: 2.3099 - val_acc
uracy: 0.4725 - lr: 2.5419e-04 - 13s/epoch - 67ms/step
Epoch 15/20
194/194 - 13s - loss: 1.8573 - accuracy: 0.5155 - val_loss: 2.2618 - val_acc
uracy: 0.4675 - lr: 2.2877e-04 - 13s/epoch - 67ms/step
```

```
Epoch 16/20
194/194 - 14s - loss: 1.7947 - accuracy: 0.5223 - val_loss: 2.2470 - val_acc
uracy: 0.4650 - lr: 2.0589e-04 - 14s/epoch - 70ms/step
Epoch 17/20
194/194 - 14s - loss: 1.7526 - accuracy: 0.5236 - val_loss: 2.2484 - val_acc
uracy: 0.4775 - lr: 1.8530e-04 - 14s/epoch - 73ms/step
Epoch 18/20
194/194 - 16s - loss: 1.7019 - accuracy: 0.5379 - val_loss: 2.1913 - val_acc
uracy: 0.4875 - lr: 1.6677e-04 - 16s/epoch - 81ms/step
Epoch 19/20
194/194 - 16s - loss: 1.6614 - accuracy: 0.5502 - val_loss: 2.2044 - val_acc
uracy: 0.4900 - lr: 1.5009e-04 - 16s/epoch - 83ms/step
Epoch 20/20
194/194 - 19s - loss: 1.6224 - accuracy: 0.5437 - val_loss: 2.1668 - val_acc
uracy: 0.4800 - lr: 1.3509e-04 - 19s/epoch - 97ms/step
```

In [56]:

```
#Plotting model accuracy and loss
from matplotlib import pyplot as plt
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





In [57]:

```
y_test.shape
```

Out[57]:

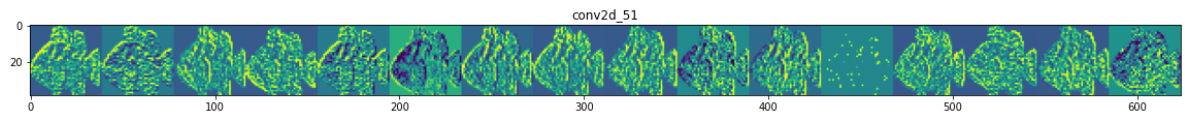
```
(1325, 193)
```

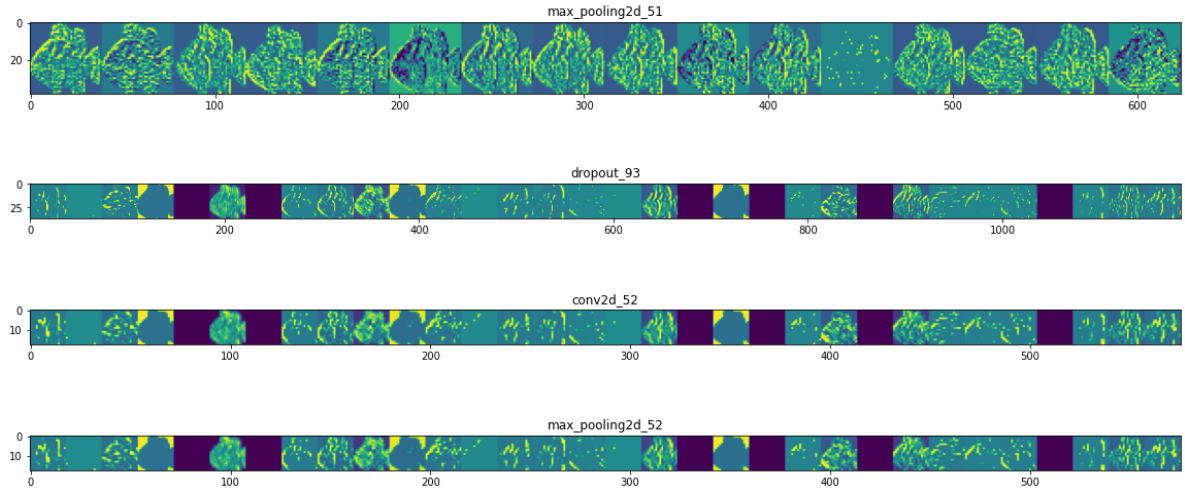
In [66]:

```
#visualization
from tensorflow.keras.preprocessing.image import img_to_array, load_img
successive_outputs = [layer.output for layer in model.layers[1:]]
visualization_model = tf.keras.models.Model(inputs = model.input, outputs = successive_outp
x = X_test
successive_feature_maps = visualization_model.predict(x)
# Retrieve are the names of the layers, so can have them as part of our plot
layer_names = [layer.name for layer in model.layers]
for layer_name, feature_map in zip(layer_names, successive_feature_maps):
  print(feature_map.shape)
  if len(feature_map.shape) == 4:

    # Plot Feature maps for the conv / maxpool layers, not the fully-connected layers

    n_features = feature_map.shape[-1]  # number of features in the feature map
    size       = feature_map.shape[ 1]  # feature map shape (1, size, size, n_features)

    # We will tile our images in this matrix
    display_grid = np.zeros((size, size * n_features))

    # Postprocess the feature to be visually palatable
    for i in range(n_features):
      x  = feature_map[0, :, :, i]
      x -= x.mean()
      x /= x.std ()
      x *=  64
      x += 128
      x  = np.clip(x, 0, 255).astype('uint8')
      # Tile each filter into a horizontal grid
      display_grid[:, i * size : (i + 1) * size] = x
# Display the grid
    scale = 20. / n_features
    plt.figure( figsize=(scale * n_features, scale) )
    plt.title ( layer_name )
    plt.grid  ( False )
    plt.imshow( display_grid, aspect='auto', cmap='viridis' )
```

```
(1325, 39, 39, 16)
(1325, 39, 39, 16)
(1325, 37, 37, 32)
(1325, 18, 18, 32)
(1325, 18, 18, 32)
(1325, 10368)
(1325, 512)
(1325, 512)
(1325, 1024)
(1325, 1024)
(1325, 193)
```

```
C:\Users\HPINFO~1\AppData\Local\Temp/ipykernel_16408/4247667521.py:38: Runti
meWarning: invalid value encountered in true_divide
  x /= x.std ()
```





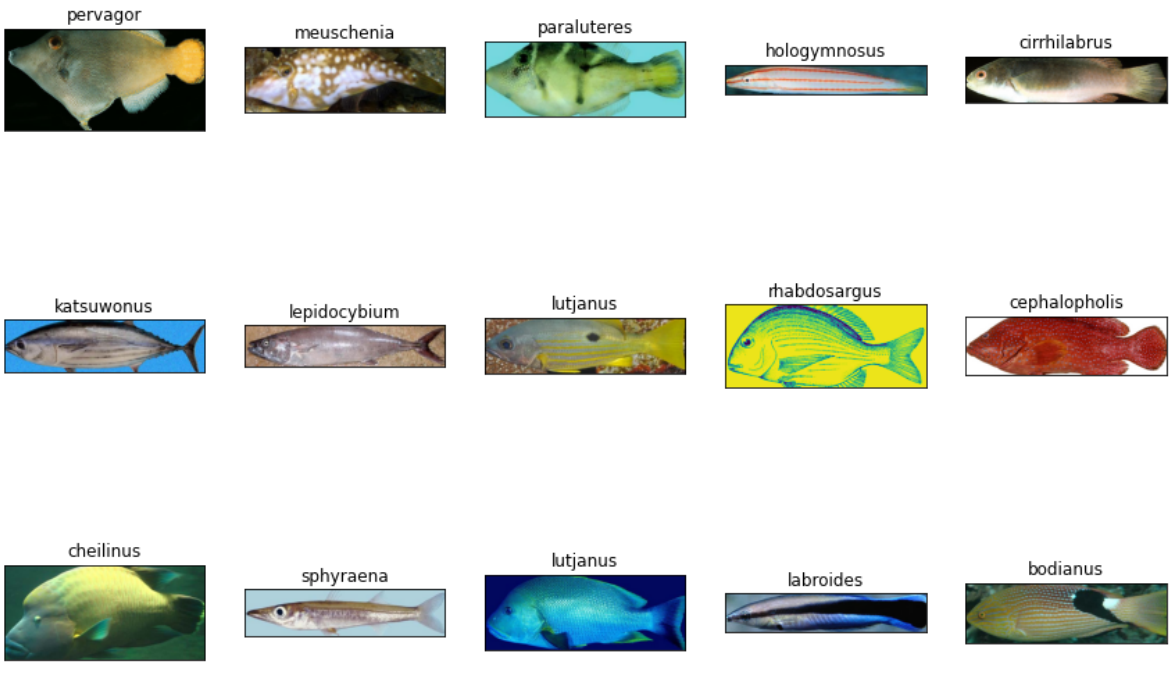
In [63]:

```
# Deriving final loss
final_loss, final_acc = model.evaluate(X_test, y_test, verbose=0)
print("Final loss: {0:.4f}, final accuracy: {1:.4f}".format(final_loss, final_acc))
```

```
Final loss: 2.0764, final accuracy: 0.5192
```

In [69]:

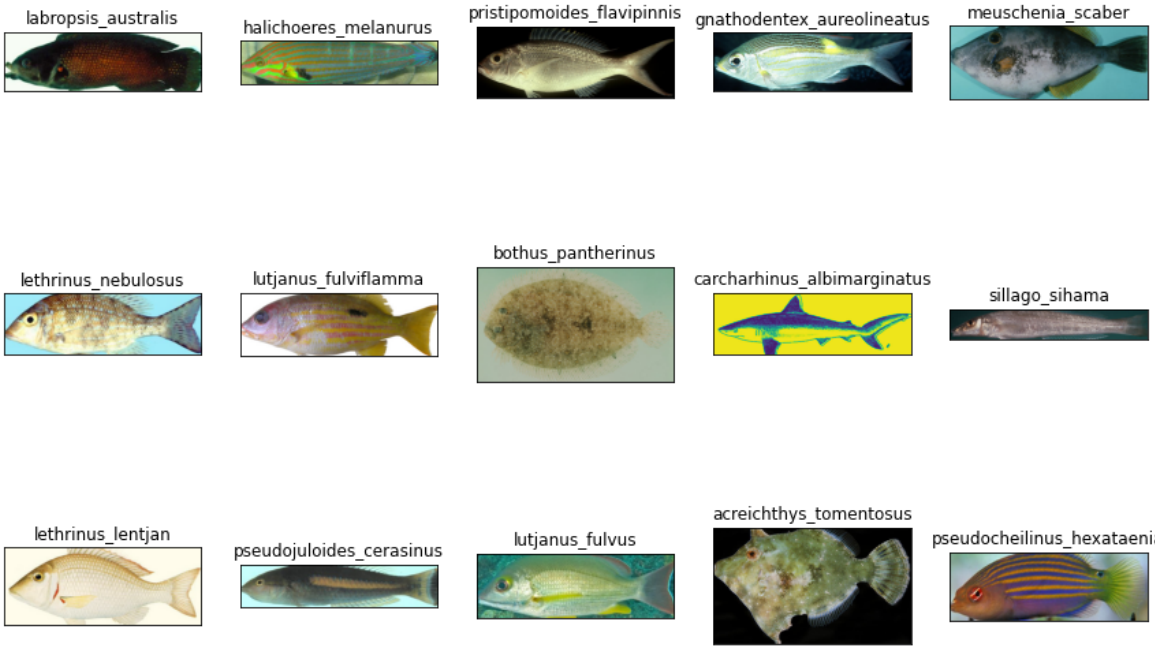
```
from pathlib import Path
import pandas as pd

image_dir = Path('C:\\Users\\HP Info\\Documents\\MACS\\Semester 3\\CSCI6515 - MLBG\\Project

# Get filepaths and labels
filepaths = list(image_dir.glob('*.png'))
labels = []
for file in filepaths:
    labels.append(os.path.split(file)[1].split("_")[0])

filepaths = pd.Series(filepaths, name='Filepaths').astype(str)
labels = pd.Series(labels, name='Labels')

# Concatenate filepaths and labels
image_df = pd.concat([filepaths, labels], axis=1)

# Drop GT images
image_df = image_df[image_df['Labels'].apply(lambda x: x[-2:] != 'GT')]

# Shuffle the DataFrame and reset index
image_df = image_df.sample(frac=1).reset_index(drop = True)

# Show the result
image_df.head()
#Displaying a subsample of the dataset
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(15,10), subplot_kw={'xticks':[], 'ytick
for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(image_df.Filepaths[i]))
    ax.set_title(image_df.Labels[i])

plt.show()
```



In [71]:

```
#For visualizing the contents of the folder
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

import os
from pathlib import Path
image_dir = Path('C:\\Users\\HP Info\\Documents\\MACS\\Semester 3\\CSCI6515 - MLBG\\Project
index_file = pd.read_csv('C:\\Users\\HP Info\\Documents\\MACS\\Semester 3\\CSCI6515 - MLBG\
print(index_file.head())

filepaths = list(image_dir.glob(r'**/*.png'))
labels = []
for filename in filepaths:
    base = os.path.basename(filename)
    name = os.path.splitext(base)[0]
    #print(name)
    labels.append(name)

#labels = list(lambda x: os.path.basename(filepaths))
#filename = os.path.basename("path/to/file/sample.txt")
#print(filename)

filepaths = pd.Series(filepaths, name='Filepaths').astype(str)
labels = pd.Series(labels, name='Species Label')

# Concatenate filepaths and labels
image_df = pd.concat([filepaths, labels], axis=1)

# Drop GT images
image_df = image_df[image_df['Species Label'].apply(lambda x: x[-2:] != 'GT')]

# Shuffle the DataFrame and reset index
image_df = image_df.sample(frac=1).reset_index(drop = True)
#image_df['Source'] = index
# Show the result
#image_df.head()
df = pd.merge(image_df, index_file,left_on="Species Label", right_on="Species Label")
df = df.drop(["Label", "Index"], axis = 1)
train_df, test_df = train_test_split(df, train_size=0.85, shuffle=True, random_state=1)
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(15,10), subplot_kw={'xticks':[], 'ytick
for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(df.Filepaths[i]))
    ax.set_title(df.Species[i])

plt.show()
```

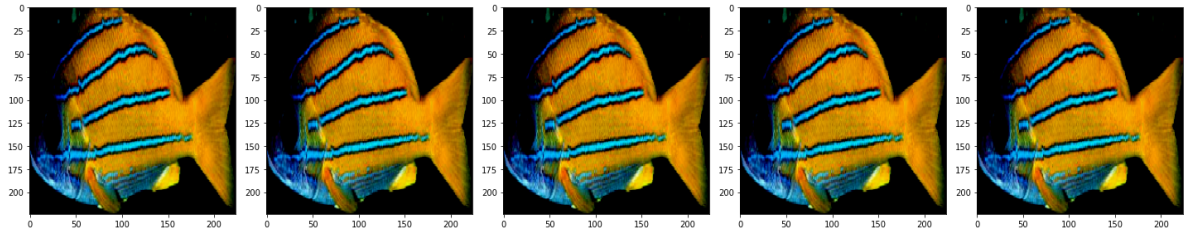
```
   Label   Species      Source Species Label  Index
0      1  A73EGS-P  controlled    A73EGS-P_1    1.0
1      1  A73EGS-P    sketches    A73EGS-P_2    2.0
2      1  A73EGS-P  controlled    A73EGS-P_3    3.0
3      1  A73EGS-P  controlled    A73EGS-P_4    4.0
4      1  A73EGS-P  controlled    A73EGS-P_5    5.0
```


labropsis_australis
halichoeres_melanurus
pristipomoides_flavipinnis
gnathodentex_aureolineatus
meuschenia_scaber
lethrinus_nebulosus
lutjanus_fulviflamma
bothus_pantherinus
carcharhinus_albimarginatus
sillago_sihama
lethrinus_lentjan
pseudojuloides_cerasinus
lutjanus_fulvus
acreichthys_tomentosus
pseudocheilinus_hexataenia

In [72]:

```
#Augmented data visualization
train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input,
    validation_split=0.15
)
train_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepaths',
    y_col='Species',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=80,
    shuffle=True,
    seed=42,
    subset='training'
)

def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip( images_arr, axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.show()

augmented_images = [train_images[0][0][0] for i in range(5)]
plotImages(augmented_images)
```

```
Found 3060 validated image filenames belonging to 459 classes.

Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



In [ ]: