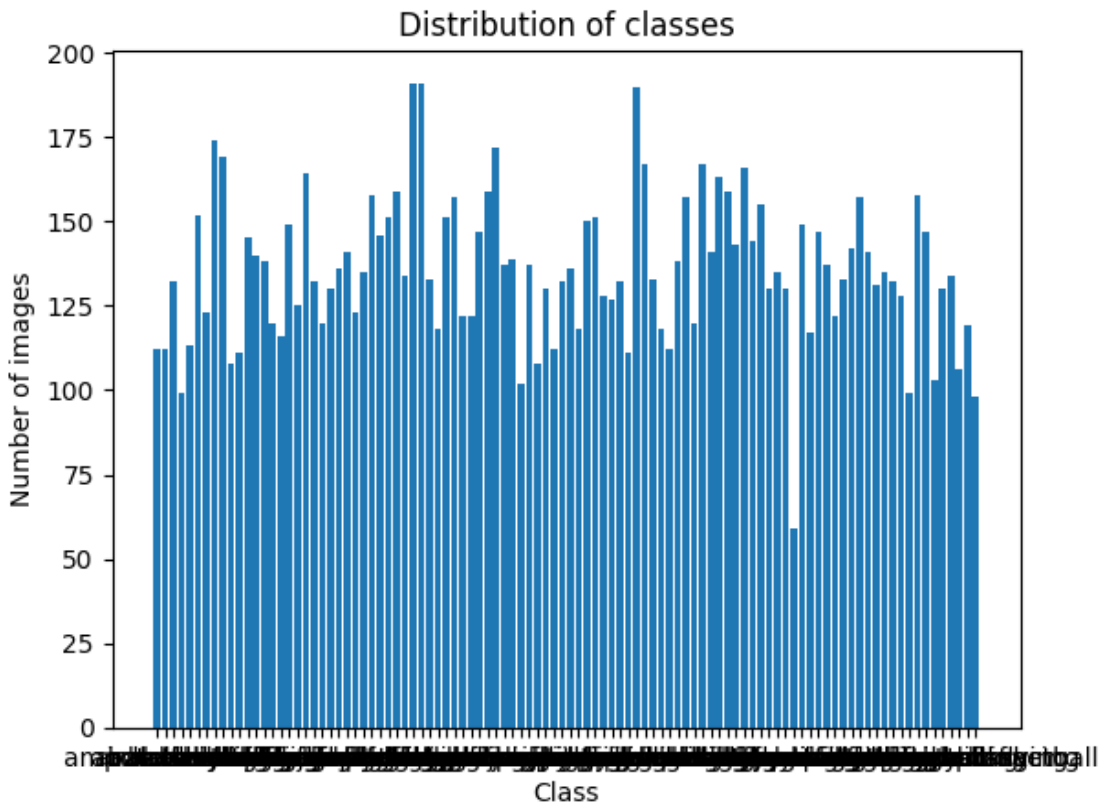
Image_Classification

April 27, 2023

[12]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
import seaborn as sb
import keras
import zipfile


test_folder= 'Downloads/sports/test'
train_folder= 'Downloads/sports/train'

subdirs=[name for name in os.listdir(train_folder) if os.path.isdir(os.path.
 ↪join(train_folder, name))]
image_counts={}
for x in subdirs:
    subdir_path = os.path.join(train_folder, x)
    if os.path.isdir(subdir_path):
        images = os.listdir(subdir_path)
        image_counts[x]=len(images)


plt.bar(image_counts.keys(), image_counts.values())

plt.xlabel('Class')
plt.ylabel('Number of images')
plt.title('Distribution of classes')
plt.show()

#This bar graph shows the amount of pictures of each sport in our target data.␣
 ↪Obviously, since there are so many sports, the labels are illegible,␣
 ↪however, the concept is still shown.
#The model should be able to predict what sport an image is of out of the given␣
 ↪categories.
```



[20]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout

# Define input shape
input_shape = (img_size, img_size, 3)

# Define model architecture
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(subdirs), activation='softmax')
])
```

```python
# Compile model
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Train model
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=test_generator,
    verbose=1
)
```

```
Epoch 1/10
425/425 [==============================] - 618s 1s/step - loss: 4.5555 -
accuracy: 0.0195 - val_loss: 4.4023 - val_accuracy: 0.0340
Epoch 2/10
425/425 [==============================] - 437s 1s/step - loss: 4.3476 -
accuracy: 0.0440 - val_loss: 4.0932 - val_accuracy: 0.0700
Epoch 3/10
425/425 [==============================] - 426s 1s/step - loss: 4.0809 -
accuracy: 0.0772 - val_loss: 3.7978 - val_accuracy: 0.1100
Epoch 4/10
425/425 [==============================] - 427s 1s/step - loss: 3.7813 -
accuracy: 0.1157 - val_loss: 3.3938 - val_accuracy: 0.1760
Epoch 5/10
425/425 [==============================] - 426s 1s/step - loss: 3.5363 -
accuracy: 0.1481 - val_loss: 2.9441 - val_accuracy: 0.2680
Epoch 6/10
425/425 [==============================] - 426s 1s/step - loss: 3.3429 -
accuracy: 0.1861 - val_loss: 2.6569 - val_accuracy: 0.3240
Epoch 7/10
425/425 [==============================] - 427s 1s/step - loss: 3.2328 -
accuracy: 0.1971 - val_loss: 2.7752 - val_accuracy: 0.2920
Epoch 8/10
425/425 [==============================] - 517s 1s/step - loss: 3.1167 -
accuracy: 0.2197 - val_loss: 2.4763 - val_accuracy: 0.3640
Epoch 9/10
425/425 [==============================] - 469s 1s/step - loss: 3.0165 -
accuracy: 0.2381 - val_loss: 2.4296 - val_accuracy: 0.3680
Epoch 10/10
425/425 [==============================] - 424s 997ms/step - loss: 2.9489 -
accuracy: 0.2496 - val_loss: 2.3062 - val_accuracy: 0.3940
```

[16]:
```
# Define input shape
input_shape = (img_size, img_size, 3)

# Define model architecture
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(subdirs), activation='softmax')
])

# Compile model
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Train model
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=test_generator,
    verbose=1
)

# Evaluate model on test data
test_loss, test_accuracy = model.evaluate(test_generator)
print(f'Test loss: {test_loss:.3f}')
print(f'Test accuracy: {test_accuracy:.3f}')
```

```
Epoch 1/10
425/425 [==============================] - 420s 985ms/step - loss: 4.5214 -
accuracy: 0.0245 - val_loss: 4.3376 - val_accuracy: 0.0520
Epoch 2/10
425/425 [==============================] - 503s 1s/step - loss: 4.2574 -
accuracy: 0.0515 - val_loss: 3.8279 - val_accuracy: 0.0980
Epoch 3/10
425/425 [==============================] - 454s 1s/step - loss: 3.9411 -
accuracy: 0.0984 - val_loss: 3.4394 - val_accuracy: 0.1900
Epoch 4/10
425/425 [==============================] - 418s 982ms/step - loss: 3.6137 -
```

```
accuracy: 0.1472 - val_loss: 2.9469 - val_accuracy: 0.2720
Epoch 5/10
425/425 [==============================] - 418s 983ms/step - loss: 3.3401 -
accuracy: 0.1842 - val_loss: 2.7669 - val_accuracy: 0.3020
Epoch 6/10
425/425 [==============================] - 417s 981ms/step - loss: 3.1602 -
accuracy: 0.2184 - val_loss: 2.5580 - val_accuracy: 0.3640
Epoch 7/10
425/425 [==============================] - 418s 984ms/step - loss: 3.0309 -
accuracy: 0.2455 - val_loss: 2.2719 - val_accuracy: 0.3780
Epoch 8/10
425/425 [==============================] - 419s 985ms/step - loss: 2.9031 -
accuracy: 0.2698 - val_loss: 2.2367 - val_accuracy: 0.4040
Epoch 9/10
425/425 [==============================] - 415s 977ms/step - loss: 2.8068 -
accuracy: 0.2867 - val_loss: 2.1389 - val_accuracy: 0.4280
Epoch 10/10
425/425 [==============================] - 414s 973ms/step - loss: 2.7205 -
accuracy: 0.3063 - val_loss: 2.0091 - val_accuracy: 0.4600
16/16 [==============================] - 4s 227ms/step - loss: 2.0091 -
accuracy: 0.4600
Test loss: 2.009
Test accuracy: 0.460
```

[21]:
```python
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.layers import GlobalAveragePooling2D

# Load pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False,␣
 ↪input_shape=input_shape)

# Freeze base layers
for layer in base_model.layers:
    layer.trainable = False

# Add custom classification head
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(len(subdirs), activation='softmax')(x)

# Combine base and custom head
model = tf.keras.models.Model(inputs=base_model.input, outputs=predictions)

# Compile model
model.compile(
```

```
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Train model with transfer learning
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=test_generator,
    verbose=1
)

# Evaluate model on test data
test_loss, test_accuracy = model.evaluate(test_generator)
print(f'Test loss: {test_loss:.3f}')
print(f'Test accuracy: {test_accuracy:.3f}')
```

```
Epoch 1/10
425/425 [==============================] - 1916s 5s/step - loss: 4.0642 -
accuracy: 0.1039 - val_loss: 3.1650 - val_accuracy: 0.2680
Epoch 2/10
425/425 [==============================] - 1948s 5s/step - loss: 2.9200 -
accuracy: 0.3064 - val_loss: 2.2703 - val_accuracy: 0.4440
Epoch 3/10
425/425 [==============================] - 1976s 5s/step - loss: 2.3727 -
accuracy: 0.4158 - val_loss: 1.8329 - val_accuracy: 0.5360
Epoch 4/10
425/425 [==============================] - 1983s 5s/step - loss: 2.0642 -
accuracy: 0.4804 - val_loss: 1.5910 - val_accuracy: 0.5860
Epoch 5/10
425/425 [==============================] - 1974s 5s/step - loss: 1.8744 -
accuracy: 0.5173 - val_loss: 1.4174 - val_accuracy: 0.6220
Epoch 6/10
425/425 [==============================] - 1965s 5s/step - loss: 1.7112 -
accuracy: 0.5531 - val_loss: 1.2682 - val_accuracy: 0.6540
Epoch 7/10
425/425 [==============================] - 1984s 5s/step - loss: 1.6031 -
accuracy: 0.5784 - val_loss: 1.2649 - val_accuracy: 0.6300
Epoch 8/10
425/425 [==============================] - 1991s 5s/step - loss: 1.5225 -
accuracy: 0.5995 - val_loss: 1.1041 - val_accuracy: 0.6960
Epoch 9/10
425/425 [==============================] - 1989s 5s/step - loss: 1.4417 -
accuracy: 0.6104 - val_loss: 1.0902 - val_accuracy: 0.6780
Epoch 10/10
425/425 [==============================] - 1963s 5s/step - loss: 1.3801 -
accuracy: 0.6264 - val_loss: 0.9924 - val_accuracy: 0.7240
```

```
16/16 [==============================] - 69s 4s/step - loss: 0.9924 - accuracy:
0.7240
Test loss: 0.992
Test accuracy: 0.724
```

[ ]:
```
#Analysis
#Obviously, a pre-trained model is going to produce the best results in this␣
 ↪scenario. However, I was suprised to see that both the sequential model as␣
 ↪well as the CNN model improved significantly and proved to also be good␣
 ↪predictors in such a large dataset.
```