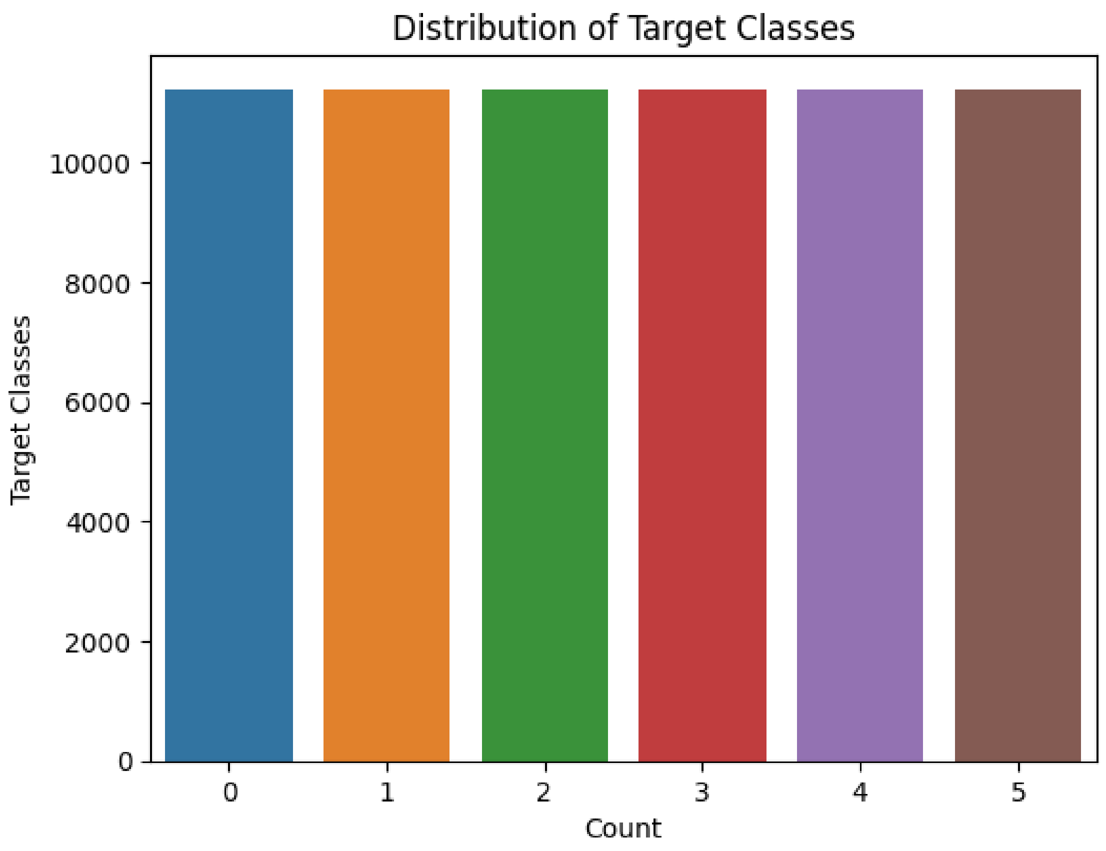
Image Classification with DL

April 14, 2023

[2]:
```
import os
import numpy as np
import cv2
import tensorflow as tf
from sklearn.model_selection import train_test_split

# Stores category subfolders that will iterate through each image
categories = ['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']
image_data = []
target_classes = []

# Loop through the categories and append the data and target lists
for category in categories:
    script_path = os.path.abspath('archive/seg_train/seg_train')
    folders = os.path.join(script_path, category)
    for filename in os.listdir(folders):
        img_path = os.path.join(folders, filename)
        # Read image data as numpy array to train model on the binary matrix␣
↪data
        img_data = cv2.imread(img_path)
        # Convert image to RGB format so we can normalize pixels later
        img_data = cv2.cvtColor(img_data, cv2.COLOR_BGR2RGB)
        # Resize image to fixed size
        img_data = cv2.resize(img_data, (150, 150))
        # Normalize the pixel values of the image data
        img_data = img_data.astype('float32')
        img_data /= 255
        image_data.append(img_data)
        target_classes.append(categories.index(category))

# Convert the image data and target classes to numpy arrays
image_data = np.array(image_data).astype('float32')
target_classes = np.array(target_classes)

# Split the dataset into 80/20 train and test set
x_train, x_test, y_train, y_test = train_test_split(image_data, target_classes,␣
↪test_size=0.2, random_state=1234)
```

[3]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Convert class vectors to binary class matrices and plot distribution
num_classes = len(categories)
y_distribution = tf.keras.utils.to_categorical(y_train, num_classes)
plt = sns.countplot(y_distribution)
plt.set_title('Distribution of Target Classes')
plt.set_xlabel('Count')
plt.set_ylabel('Target Classes')
```

[3]:
```
Text(0, 0.5, 'Target Classes')
```



This data set is one of the most popular image classification datasets on Kaggle called Intel Image Classification. It contains 25,000 images of natural scenes around the worlds which are split into 6 different categories as various folders. These are named buildings, forest, glacier, mountain, sea, and street. The model should be able to classify images according to their respective category.

[5]:
```python
# Variables to use in model
batch_size = 128
```

```
num_classes = len(categories)
epochs = 20
input_shape = x_train[0].shape

print(x_train.shape, 'train samples')
print(x_test.shape, 'test samples')

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=input_shape),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(num_classes, activation='softmax'),
])
model.summary()

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

y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)

# Fit the model to the training data
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
(11227, 150, 150, 3) train samples
(2807, 150, 150, 3) test samples
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 67500)             0

 dense_3 (Dense)             (None, 512)               34560512

 dropout_2 (Dropout)         (None, 512)               0
 dense_4 (Dense)             (None, 512)              262656

 dropout_3 (Dropout)         (None, 512)              0

 dense_5 (Dense)             (None, 6)                3078

=================================================================
Total params: 34,826,246
Trainable params: 34,826,246
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
88/88 [==============================] - 17s 170ms/step - loss: 17.6338 -
accuracy: 0.2123 - val_loss: 1.7290 - val_accuracy: 0.2216
Epoch 2/20
88/88 [==============================] - 13s 145ms/step - loss: 2.1144 -
accuracy: 0.1986 - val_loss: 1.7797 - val_accuracy: 0.1803
Epoch 3/20
88/88 [==============================] - 12s 137ms/step - loss: 1.8062 -
accuracy: 0.2669 - val_loss: 1.6710 - val_accuracy: 0.3035
Epoch 4/20
88/88 [==============================] - 12s 132ms/step - loss: 1.8044 -
accuracy: 0.2750 - val_loss: 1.6535 - val_accuracy: 0.2953
Epoch 5/20
88/88 [==============================] - 11s 128ms/step - loss: 1.6902 -
accuracy: 0.3122 - val_loss: 1.5782 - val_accuracy: 0.3495
Epoch 6/20
88/88 [==============================] - 11s 129ms/step - loss: 1.7482 -
accuracy: 0.3312 - val_loss: 1.5763 - val_accuracy: 0.3402
Epoch 7/20
88/88 [==============================] - 13s 145ms/step - loss: 1.6386 -
accuracy: 0.3383 - val_loss: 1.5862 - val_accuracy: 0.3363
Epoch 8/20
88/88 [==============================] - 12s 138ms/step - loss: 1.6058 -
accuracy: 0.3476 - val_loss: 4.4834 - val_accuracy: 0.1820
Epoch 9/20
88/88 [==============================] - 12s 140ms/step - loss: 1.6293 -
accuracy: 0.3433 - val_loss: 1.5179 - val_accuracy: 0.3837
Epoch 10/20
88/88 [==============================] - 13s 150ms/step - loss: 1.5934 -
accuracy: 0.3542 - val_loss: 1.5459 - val_accuracy: 0.3659
Epoch 11/20
88/88 [==============================] - 13s 141ms/step - loss: 1.5714 -
accuracy: 0.3644 - val_loss: 1.6250 - val_accuracy: 0.3359
Epoch 12/20
88/88 [==============================] - 12s 132ms/step - loss: 1.5697 -
accuracy: 0.3588 - val_loss: 1.5879 - val_accuracy: 0.3926
```

```
Epoch 13/20
88/88 [==============================] - 12s 132ms/step - loss: 1.5607 -
accuracy: 0.3834 - val_loss: 1.4628 - val_accuracy: 0.4264
Epoch 14/20
88/88 [==============================] - 12s 141ms/step - loss: 1.5520 -
accuracy: 0.3906 - val_loss: 1.4677 - val_accuracy: 0.4118
Epoch 15/20
88/88 [==============================] - 12s 135ms/step - loss: 1.5524 -
accuracy: 0.3916 - val_loss: 1.4148 - val_accuracy: 0.4524
Epoch 16/20
88/88 [==============================] - 12s 136ms/step - loss: 1.5282 -
accuracy: 0.4001 - val_loss: 1.4186 - val_accuracy: 0.4428
Epoch 17/20
88/88 [==============================] - 12s 140ms/step - loss: 1.5133 -
accuracy: 0.4052 - val_loss: 1.4842 - val_accuracy: 0.4129
Epoch 18/20
88/88 [==============================] - 12s 137ms/step - loss: 1.5051 -
accuracy: 0.4122 - val_loss: 1.3937 - val_accuracy: 0.4517
Epoch 19/20
88/88 [==============================] - 12s 133ms/step - loss: 1.4817 -
accuracy: 0.4164 - val_loss: 1.4885 - val_accuracy: 0.3766
Epoch 20/20
88/88 [==============================] - 12s 135ms/step - loss: 1.4627 -
accuracy: 0.4212 - val_loss: 1.5445 - val_accuracy: 0.3833
88/88 [==============================] - 3s 27ms/step - loss: 1.5445 - accuracy:
0.3833
Test loss: 1.5445009469985962
Test accuracy: 0.3833274245262146
```

[6]:
```
# Replace Sequential model with CNN model
model = tf.keras.models.Sequential([
  tf.keras.Input(shape=input_shape),
  tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
  tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
  tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
  tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Dense(num_classes, activation="softmax"),
])
model.summary()

# Compile model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))

# Evaluate model on the test data
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 74, 74, 32)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 72, 72, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 36, 36, 64)       0         
 2D)                                                             
                                                                 
 flatten_2 (Flatten)         (None, 82944)             0         
                                                                 
 dropout_4 (Dropout)         (None, 82944)             0         
                                                                 
 dense_6 (Dense)             (None, 6)                 497670    
                                                                 
=================================================================
Total params: 517,062
Trainable params: 517,062
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
88/88 [==============================] - 21s 220ms/step - loss: 1.2237 -
accuracy: 0.5431 - val_loss: 0.8791 - val_accuracy: 0.6591
Epoch 2/20
88/88 [==============================] - 17s 196ms/step - loss: 0.7751 -
accuracy: 0.7141 - val_loss: 0.7546 - val_accuracy: 0.7175
Epoch 3/20
88/88 [==============================] - 17s 197ms/step - loss: 0.6548 -
accuracy: 0.7643 - val_loss: 0.6542 - val_accuracy: 0.7663
Epoch 4/20
88/88 [==============================] - 18s 205ms/step - loss: 0.5802 -
accuracy: 0.7913 - val_loss: 0.6390 - val_accuracy: 0.7756
Epoch 5/20
```

```
88/88 [==============================] - 18s 204ms/step - loss: 0.4975 -
accuracy: 0.8293 - val_loss: 0.6819 - val_accuracy: 0.7567
Epoch 6/20
88/88 [==============================] - 18s 201ms/step - loss: 0.4394 -
accuracy: 0.8457 - val_loss: 0.7748 - val_accuracy: 0.7403
Epoch 7/20
88/88 [==============================] - 16s 186ms/step - loss: 0.3861 -
accuracy: 0.8680 - val_loss: 0.6273 - val_accuracy: 0.7813
Epoch 8/20
88/88 [==============================] - 22s 254ms/step - loss: 0.3191 -
accuracy: 0.8932 - val_loss: 0.6708 - val_accuracy: 0.7784
Epoch 9/20
88/88 [==============================] - 23s 257ms/step - loss: 0.2674 -
accuracy: 0.9137 - val_loss: 0.6473 - val_accuracy: 0.7852
Epoch 10/20
88/88 [==============================] - 17s 197ms/step - loss: 0.2242 -
accuracy: 0.9252 - val_loss: 0.7249 - val_accuracy: 0.7617
Epoch 11/20
88/88 [==============================] - 18s 201ms/step - loss: 0.2114 -
accuracy: 0.9287 - val_loss: 0.7688 - val_accuracy: 0.7545
Epoch 12/20
88/88 [==============================] - 17s 192ms/step - loss: 0.1726 -
accuracy: 0.9436 - val_loss: 0.7645 - val_accuracy: 0.7841
Epoch 13/20
88/88 [==============================] - 16s 186ms/step - loss: 0.1582 -
accuracy: 0.9493 - val_loss: 0.7969 - val_accuracy: 0.7773
Epoch 14/20
88/88 [==============================] - 16s 187ms/step - loss: 0.1409 -
accuracy: 0.9560 - val_loss: 0.8099 - val_accuracy: 0.7727
Epoch 15/20
88/88 [==============================] - 18s 209ms/step - loss: 0.1163 -
accuracy: 0.9642 - val_loss: 0.8604 - val_accuracy: 0.7699
Epoch 16/20
88/88 [==============================] - 18s 204ms/step - loss: 0.1136 -
accuracy: 0.9651 - val_loss: 0.8933 - val_accuracy: 0.7688
Epoch 17/20
88/88 [==============================] - 17s 199ms/step - loss: 0.1111 -
accuracy: 0.9636 - val_loss: 0.9461 - val_accuracy: 0.7702
Epoch 18/20
88/88 [==============================] - 17s 197ms/step - loss: 0.0893 -
accuracy: 0.9737 - val_loss: 0.9121 - val_accuracy: 0.7809
Epoch 19/20
88/88 [==============================] - 17s 191ms/step - loss: 0.0845 -
accuracy: 0.9739 - val_loss: 0.9781 - val_accuracy: 0.7752
Epoch 20/20
88/88 [==============================] - 17s 197ms/step - loss: 0.0776 -
accuracy: 0.9763 - val_loss: 0.9813 - val_accuracy: 0.7759
Test loss: 0.981346845626831
Test accuracy: 0.7759174108505249
```

The next part of the assignment asked us to practice transfer learning and I decided to do this using a pre-trained model from the Tensorflow Hub. The link for the tutorial used to generate the code can be found here. However, I experimented with this in a seperate Colab Notebook.

In terms of performance, the CNN model outperformed the sequential model in test loss and accuracy. The sequential model had a test loss of 1.5445009469985962 and a test accuracy of 0.3833274245262146, which suggests the model could not learn the relationships of the different images within the dataset very well and made poor predictions when trying to classify the test data. However, the CNN model had a test loss of 0.981346845626831 and a test accuracy of 0.7759174108505249, which suggests the model was able to learn the relationships of the different images within the dataset significantly better than the sequential model. In conclusion, the results of this analysis demonstrate the superior performance of the CNN model over the sequential model in the image classification task. A CNN model is a good choice for image classification, and choosing the wrong model can mean much lower performance.