Homework 1 Deep Learnin - ELE2765

Nome: Thiago Matheus Bruno da Silva	Matrícula: 1413286	Data: 29/12/2006
Capítulo: Scene Classification of Remote Sensing Images		Parte: 1

Objective: Implement inKerasand tune a deep neural network for scene classification.

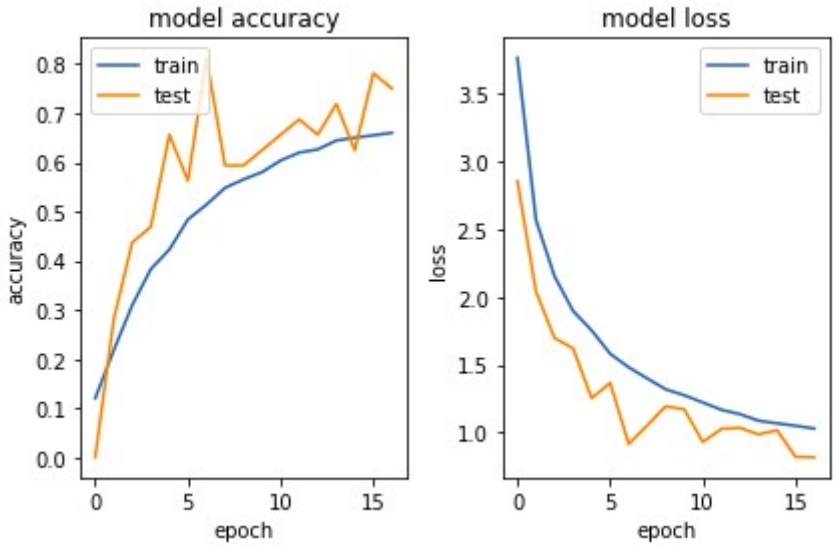


Illustration 1: Modelo 1

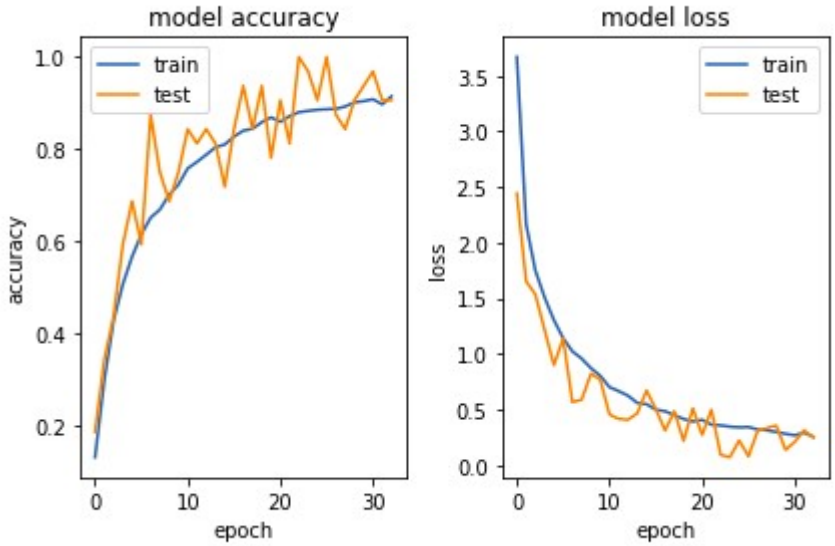


Illustration 2: Modelo 2

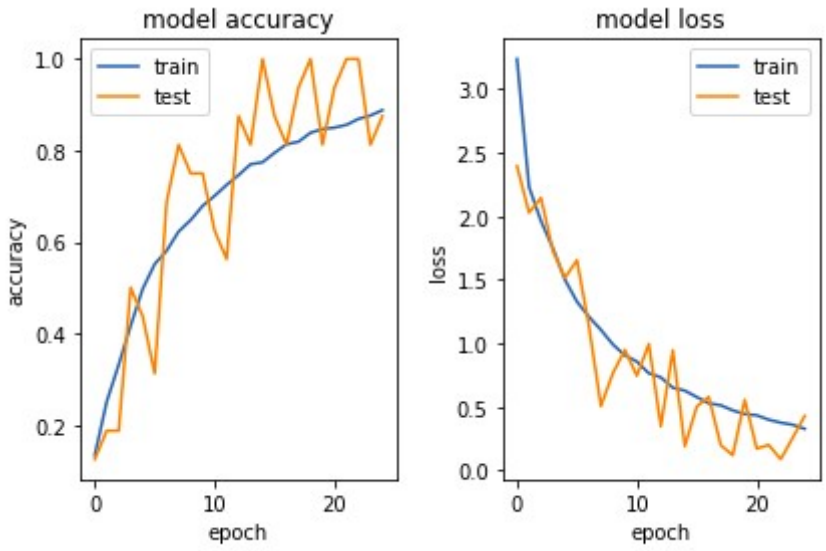


Illustration 3: Modelo 3

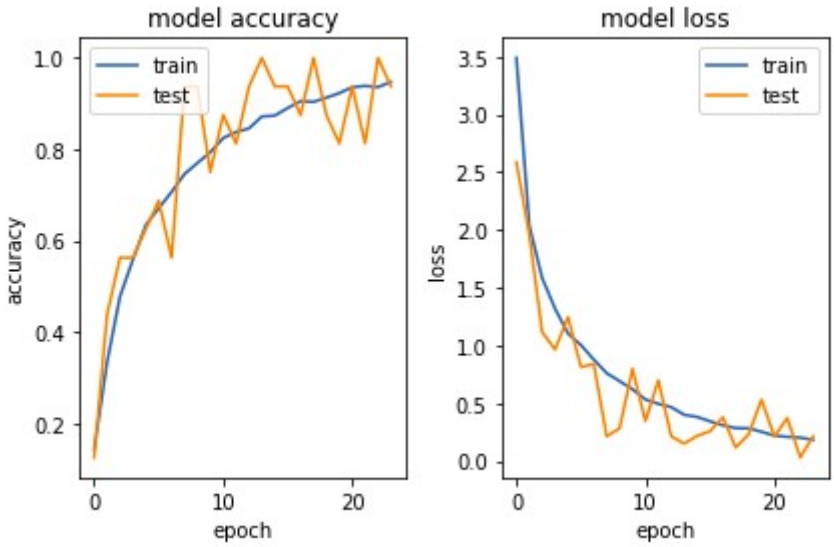


Illustration 4: Modelo 4

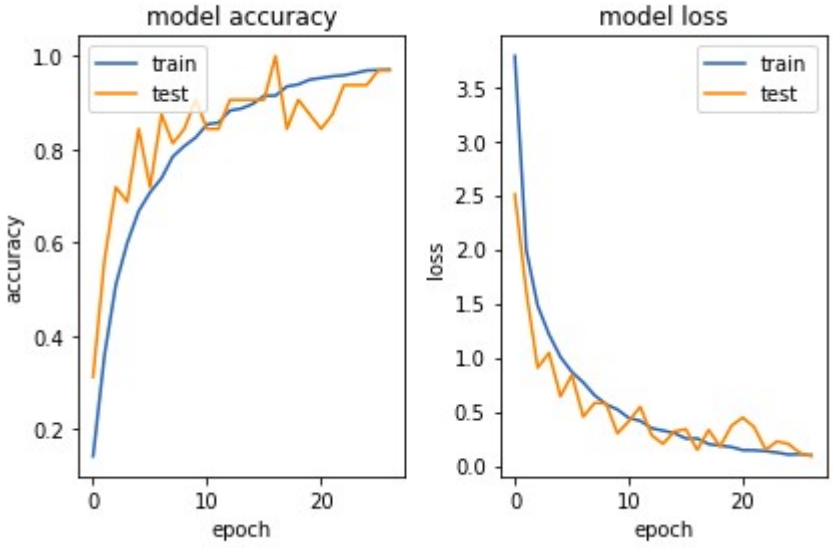


Illustration 5: Modelo 5

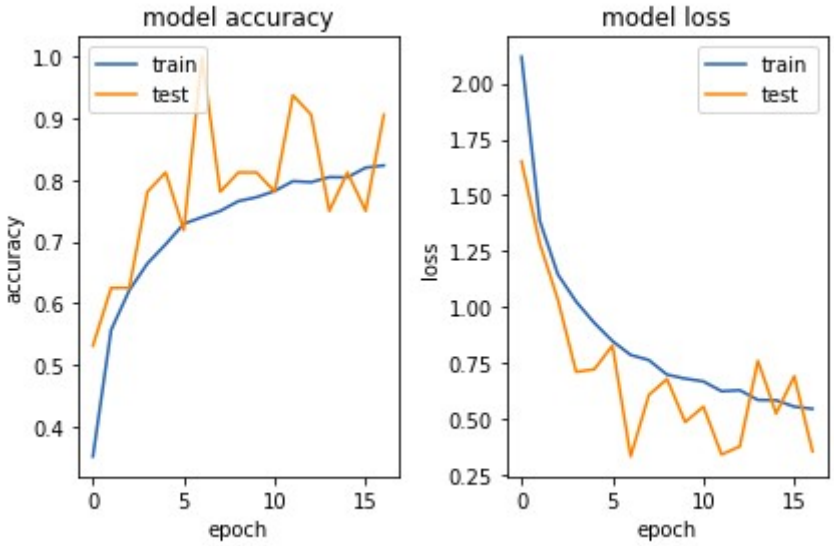


Illustration 6: Modelo 6

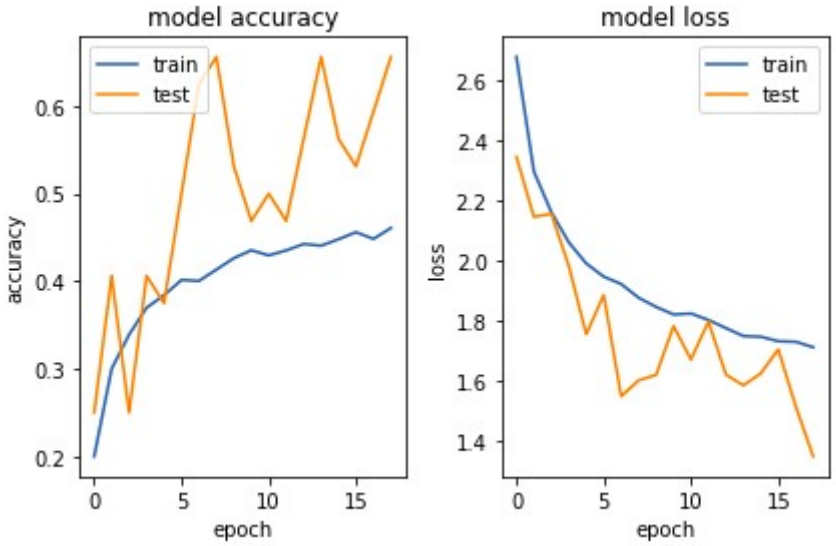


Illustration 7: Modelo 7

Modelos/Parâmetros	Arquitetura	Augmentations	Batch-size	Learning rate	Decay	Test Accuracy	Test Loss
Modelo 1	VGG11	Sim	32	1e-3	0.1	0.6033	1.1029
Modelo 2	**VGG11**	**Sim**	**32**	**1e-3**	**0.01**	**0.9173**	**0.3156**
Modelo 3	VGG11	Sim	16	1e-3	0.001	0.8759	0.5683
Modelo 4	**VGG11**	**Não**	**16**	**1e-3**	**0.01**	**0.8820**	**0.2451**
Modelo 5	VGG11	Não	32	1e-3	0.01	0.8884	0.5330
Modelo 6	VGG11	Sim	32	1e-4	0.01	0.8966	0.4617
Modelo 7	VGG11	Sim	32	1e-5	0.01	0.5320	1.5608

Resolvi fazer vários testes com algumas arquiteturas, e vários hiperparâmetros. Assim, selecionei os modelos que melhores performaram e alguns que ajudaram entender o caminho até eles.

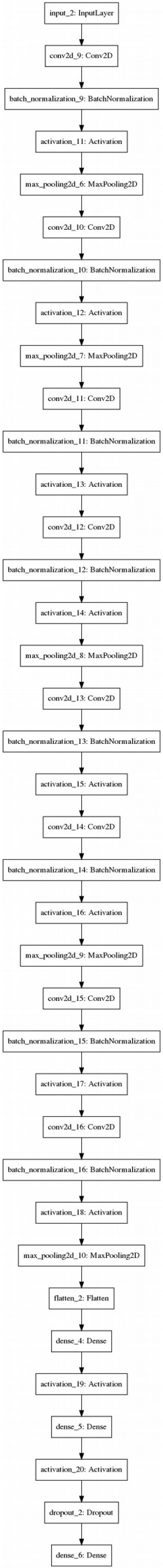
1. Arquitetura

Testei várias arquiteturas, todas baseados no moldes da VGG. Testei várias modificações com muitos parâmetros, em torno dos 20 kk, e poucos em torno dos 6 kk. Entretanto, nenhuma delas se performava muito bem. A rede com mais parâmetros não chegava aos 60%, 70% de acurácia mesmo variando os learning rates. Já aquela com poucos parâmetros conseguia ter um desenvolvimento razoável e apresentava overfitting perto dos 60%, 70%. Com menos parâmetros a rede apresentava um crescimento melhor, o que faz sentido devido ao tamanho do nosso dataset.

Por fim, acabei optando por uma rede com total de parâmetros intermediária, foi quando escolhi a VGG11. Sendo assim, como a VGG11, possui um número grande de parâmetros, precisei modificá-la um pouco. Logo, dividi o número de filtros convolucionais em cada camada por 2, fazendo assim variar de 32 até 256 e não mais de 64 até 512. Assim, diminuiríamos um dos canais do nosso tensor que chega à nossa camada classificadora MLP, diminuindo o vetor que sofreu *flatten.* Outro fator importante da VGG11, foram a quantidade de filtros por camada e a quantidade de camadas. Definir a quantidade de camadas da rede, foi essencial pois ao final de cada uma possuimos uma de max-pooling, dividindo pela metade do tamanho da nossa imagem e portanto do tensor. Além disso, escolhendo o número certo de camadas convolucionais também diminui o número total de parâmetros. Todas essas mudanças feitas ,em relação à outras redes que testei anteriormente (Menos “profundas” e, portanto, com maior quantidade de parâmetros), me ajudou a decidir qual seria a melhor arquitetura para o meu problema e como chegar em tal solução. Essa padronização da VGG de camadas convolucionais seguidas de max-pooling e a fácil manipulação dos filtros para os ajustes dos parâmetros a tornaram a escolha preferida em relação a outras redes como ResNet ou Inception por exemplo, que são um pouco mais complexas para implementar. Outra opção poderia ter sido também a AlexNet ou LeNet, porém ambas possuem grande número de parâmetros e como já expliquei seria um problema. Além disso, a LeNet não possui muitas camadas escondidas, o que dificulta o aprendizado de detalhes menores na imagem e deixar a rede mais pesada. A AlexNet por sua vez, não utiliza somente filtros 3x3, que são menos custosos computacionalmente que filtros maiores 11x11 ou 5x5. Ela também possui grande quantidade de kernels utilizados logo no começo da rede, ou seja, a imagem ainda não foi reduzida pelos max-poolings e, portanto, aumentará o custo computacional do treiamento. Já a VGG por sua vez aumenta o número de kernels gradiatavemnte com a profundidade da rede, tornando-a assim menos custosa e a melhor escolha para o problema.

Por fim, na camada de classificação, alterei de 4096 neurônios nas camadas *fully-connected* para 512. Sendo assim, deixei a rede com um total de 11kk de parâmetros aproximadamente, o que parecia adequado para o tamanho do dataset utilizado de 7k de imagens aproximadamente, de tamanho 256x256.

Além disso, adicionei antes de cada função de ativação na parte de *feature extractor* uma camada de *BatchNormalization*, com momemtum de 0.8, possibilitando teoricamente usar learning rates mais altos. Outra modificação, foi um uma camada de *Dropout* com taxa de 0.2, logo antes da camada de classificação da *softmax*. Diminui a taxa do *Dropout*, pois dá muito ruído na rede.


input_2: InputLayer
conv2d_9: Conv2D
batch_normalization_9: BatchNormalization
activation_11: Activation
max_pooling2d_6: MaxPooling2D
conv2d_10: Conv2D
batch_normalization_10: BatchNormalization
activation_12: Activation
max_pooling2d_7: MaxPooling2D
conv2d_11: Conv2D
batch_normalization_11: BatchNormalization
activation_13: Activation
conv2d_12: Conv2D
batch_normalization_12: BatchNormalization
activation_14: Activation
max_pooling2d_8: MaxPooling2D
conv2d_13: Conv2D
batch_normalization_13: BatchNormalization
activation_15: Activation
conv2d_14: Conv2D
batch_normalization_14: BatchNormalization
activation_16: Activation
max_pooling2d_9: MaxPooling2D
conv2d_15: Conv2D
batch_normalization_15: BatchNormalization
activation_17: Activation
conv2d_16: Conv2D
batch_normalization_16: BatchNormalization
activation_18: Activation
max_pooling2d_10: MaxPooling2D
flatten_2: Flatten
dense_4: Dense
activation_19: Activation
dense_5: Dense
activation_20: Activation
dropout_2: Dropout
dense_6: Dense

2. Dataset

O Dataset escolhido foi aquele com aproximadamente 7k de imagens. Este, que é uma versão já aumentada do dataset original, com aproximadaemente 1,5k imagens. Tentei usar o dataset menor e eu mesmo praticar data augmentation, porém não obtive sucesso em nenhum resultado. Sendo assim, fiz alguns testes utilizando o dataset normal, já com as augmentations, e também realizar mais data augmentation nele. Sendo assim, os que tiverem marcado na tabela, indicam o dataset de tamanho 7k e mais algumas augmentations que estarão indicadas no código comentado abaixo. As que não estão marcadas, indicam apenas a utilizaão do dataset de 7k imagens.. Todas as imagens sofreram normalização.

Para utilizar data agumentation, utilizei o Data Generator do keras, e para isso tive que fazer um reagrupamento das imagens em pastas de cada classe respectivamente.

3. Hiperparâmetros

Acredito que o ajuste dos hiperparâmetros foi uma das partes mais dificeis. Até, encontrar arquitetura correta, não conseguia encontrar um learning rate adequado. No geral, os learning rates pequenos ficavam estagnados em algum minimo local e não conseguia andar com o gradiente. Além disso, o número de batchs também ajudou bastante. No começo, escolhi usar bacthes grandes, mas os resultados também não foram muito bons. Após algumas tentativas utilizando esses hiperparâmetros juntos, percebi que ou a rede ficava estagnada ou ela crescia muito rápido, oscilando bastante também na validação. Então, inseri outro hiperparâmetro o decaimento do learning rate. Após a escolha da VGG11, comecei treinando com os parâmetros do modelo 7. Mesmo não ficando bom, percebi que a curva de validação estava crescendo na direção do modelo, além de não ter crescido muito a acurácia. Aumentando o learning rate obtive um pouco de melhora no treino como podemos ver no Modelo 6. Logo, aumentei mais um pouco learning rate, que nos leva ao Modelo 1. De novo, faz sentido a melhora com aumento do learning rate visto que foi adicionado camadas de BatchNormalization às camadas convolutivas. De um modelo para o outro eu aumentei a taxa de decaimento, pensando que o learning rate estaria muito alto durante as épocas finais do treinamento fazendo o gradiente dar saltos muito grandes numa etapa que precisava de ajustes pequenos para em encontrar o mínimo desejado. O resultado foi entretanto pior que o anterior no dataset de test, porém o gráfico de treino parecia estar convergind melhor apesar de alguns picos na acurácia. Acredito que quedas bruscar no learning rate podem ter causado ruído no modelo e tê-lo prejudicado. Assim, decidir diminuir a taxa de decaimento e obtive o Modelo 2, que foi um dos melhores resultados obtidos. Até então, estava usando o dataset com mais augmentations impostas por mim, então decidi usar o dataset original para comparar os dois resultados. Entretanto, o resultado não ficou imediatamente bom. Apenas mudando o bacth para 16, consegui um resultado parecido com o Modelo 2, e agora obtendo o Modelo 4. Sendo assim, voltando para o dataset com data augmentation, decidi diminuir ainda mais a taxa de decaimento. Porém, também não obtive resultados muito bons e somente mudando o batch-size, de novo, conseguindo o Modelo 3. Porém, se observarmos o dados de test, podemos ver que a loss ainda ficou relativamente alta, 0.5, enquanto no treino parece ter diminuido, demonstrando um pouco de overfitting. Por fim, o Modelo 5 foi obtido utilizando o dataset sem augmentations, porém, agora com um batch-size de 32. O resultado ainda foi bom, porém igual ao modelo citado anteriormente. Mesmo apresentando uma curva relativamente estável, na hora dos testes, apesar de uma acurácia alta, a loss ficou um pouco mais alta que o final do treino e validação, mostrando um possível overfitting da rede. As redes que apresentaram melhor comportamento durante o treinamento e nos testes foram os Modelos 2 e 4.

Em todos os modelos foi usado Early Stopping de 10 épocas monitorando a acurácia da validação e Save Checkpoint, para evitar overfitting e salvar sempre o melhor modelo.

```
# In[1]:

from keras.layers import Input, Dense, Conv2D, MaxPool2D, Flatten, Dropout, BatchNormalization, Activation
from keras.optimizers import Adam
from keras.models import Model
from keras.callbacks import ModelCheckpoint, EarlyStopping
import keras
import matplotlib.pyplot as plt
import numpy as np
from keras.preprocessing.image import ImageDataGenerator

import os
from tqdm import tqdm
import json


# In[10]:


def VGG_11(input_shape, classes):
    activation_f = 'relu'
    momemtum = 0.8
    input_img = Input(shape=input_shape)
    # Block 1
    # layer 1
    x = Conv2D(32, (3,3), strides=1, activation='linear', padding='same')(input_img)
    x = BatchNormalization(momentum=momemtum)(x)
    x = Activation(activation_f)(x)
    x = MaxPool2D(pool_size=(2,2), strides=2, padding='same')(x)
    # Block 2
    #layer 2
    x = Conv2D(64, (3,3), strides=1, activation='linear', padding='same')(x)
    x = BatchNormalization(momentum=momemtum)(x)
    x = Activation(activation_f)(x)
    x = MaxPool2D(pool_size=(2,2), strides=2, padding='same')(x)
    # Block 3
    #layer 3
    x = Conv2D(128, (3,3), strides=1, activation='linear', padding='same')(x)
    x = BatchNormalization(momentum=momemtum)(x)
    x = Activation(activation_f)(x)
     #layer 4
    x = Conv2D(128, (3,3), strides=1, activation='linear', padding='same')(x)
    x = BatchNormalization(momentum=momemtum)(x)
    x = Activation(activation_f)(x)
    x = MaxPool2D(pool_size=(2,2), strides=2, padding='same')(x)
    # Block 4
    #layer 5
    x = Conv2D(256, (3,3), strides=1, activation='linear', padding='same')(x)
    x = BatchNormalization(momentum=momemtum)(x)
    x = Activation(activation_f)(x)
     #layer 6
    x = Conv2D(256, (3,3), strides=1, activation='linear', padding='same')(x)
    x = BatchNormalization(momentum=momemtum)(x)
    x = Activation(activation_f)(x)
    x = MaxPool2D(pool_size=(2,2), strides=2, padding='same')(x)
    # Block 5
    # layer 7
    x = Conv2D(256, (3,3), strides=1, activation='linear', padding='same')(x)
    x = BatchNormalization(momentum=momemtum)(x)
```

```
    x = Activation(activation_f)(x)
     #layer 8
    x = Conv2D(256, (3,3), strides=1, activation='linear', padding='same')(x)
    x = BatchNormalization(momentum=momemtum)(x)
    x = Activation(activation_f)(x)
    x = MaxPool2D(pool_size=(2,2), strides=2, padding='same')(x)

    #layer 9
    #fully-connected layer
    x = Flatten()(x)

     #layer 10
    x = Dense(512, activation='linear')(x)
    x = Activation(activation_f)(x)
     #layer 11
    x = Dense(512, activation='linear')(x)
    x = Activation(activation_f)(x)
    x = Dropout(0.2)(x)

    # softmax layer (output)
    pred = Dense(classes, activation='softmax')(x)

    model  = Model(input_img , pred)

    return model



# In[3]:


def graph_training_history(history):
    acc_train = history['accuracy']
    acc_test = history['val_accuracy']
    loss_train = history['loss']
    loss_test = history['val_loss']
    #print(acc_train, acc_test, loss_train, loss_test)
    plt.rcParams['axes.facecolor']='white'
    plt.figure(1)

    # summarize history for accuracy
    plt.subplot(121)
    plt.plot(acc_train)
    plt.plot(acc_test)
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.tight_layout()

    # summarize history for loss
    plt.subplot(122)
    plt.plot(loss_train)
    plt.plot(loss_test)
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper right')
    plt.tight_layout()

    plt.show()
```

In[4]:

```
# Augmentations
train_aug=ImageDataGenerator(rescale=1./255,
                height_shift_range=0.2, featurewise_center=0,
                horizontal_flip=True, vertical_flip=True,
                zoom_range=0.3)

# No Augmentations
#train_aug = ImageDataGenerator(rescale=1./255)

#Paths dos datasets
train_folder = "UCMERCED_HW_SceneClassification/data/separated_in_labels_aug/train"
val_folder = "UCMERCED_HW_SceneClassification/data/separated_in_labels_aug/val"

# train_folder = "UCMERCED_HW_SceneClassification/data/separated_in_labels/train"
# val_folder = "UCMERCED_HW_SceneClassification/data/separated_in_labels/val"
```

In[14]:

```
# Definina o batch-size e tamanho da imagem de entrada
batch_size = 32
input_shape = (256, 256, 3)
(w, h, _) = input_shape
print(w,h)
train_generator = train_aug.flow_from_directory(train_folder,
                                  target_size=(w, h),batch_size=batch_size,
                                  class_mode='categorical', shuffle=True, seed=42)

val_generator = train_aug.flow_from_directory(val_folder,
                                target_size=(w, h),  batch_size=batch_size,
                                class_mode='categorical', shuffle=True, seed=42)
```

In[16]:

```
# Pega o tamanho total de imagens no dataset
filenames = train_generator.filenames
samples = len(filenames)
print(samples)
```

In[17]:

```
# Define o Early Stopping e salva o melhor modelo durante o treino
file_name = 'best_model.h5'
checkpointer = ModelCheckpoint(file_name, monitor='val_accuracy', save_best_only=True)
early_stop = EarlyStopping(monitor = 'val_accuracy', min_delta = 0.001,
mode = 'max', patience = 10)
callbacks=[checkpointer,early_stop]
```

In[18]:

```
# Define o número de classes, número de épocas e o learning rate e o decaimento usado
classes = 21
epochs = 100
steps_in_epoch = samples // batch_size
lr = 1e-3
adam = Adam(learning_rate = lr, decay=0.01)
net = VGG_11(input_shape, classes)
net.compile(loss = 'categorical_crossentropy', optimizer=adam , metrics=['accuracy'])
net.summary()

history = net.fit_generator(train_generator, steps_per_epoch=steps_in_epoch, epochs=epochs,
                            validation_data=val_generator,  validation_steps=1,
                            verbose=1,callbacks=callbacks)
```

In[19]:

```
# Mostra o gráfico da história do treinamento ao fim do treino
graph_training_history(history.history)
```

In[20]:

```
# Salva o modelo ao final do treino (não o melhor) e salva a história do treinamento em formato de string
net.save('weights/model_END.h5')
# Saving history
with open('weights/history_model.json', 'w') as f:
    json.dump(str(net.history.history), f)
```