TP1 IA cats and dogs

Antoine Renciot

Modèles et paramètres

Pour classifier des images de chiens et de chats, cette classification va être réalisé en CNN.
On utilise un dataset de 25 000 images.
J'ai réalisé deux modèle, un modèle avec 32 + 64 + 128 couches

```
model1 = Sequential()

model1.add(Conv2D(32, (3, 3), activation=param1['activation'],
input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS)))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Dropout(0.25))

model1.add(Conv2D(64, (3, 3), activation=param1['activation']))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Dropout(0.25))

model1.add(Conv2D(128, (3, 3), activation=param1['activation']))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Dropout(0.25))

model1.add(Flatten())
model1.add(Dense(512, activation=param1['activation']))
model1.add(BatchNormalization())
model1.add(Dropout(0.5))
model1.add(Dense(2, activation='softmax')) # 2 because we have cat and
dog classes

model1.compile(loss=param1['loss'],
optimizer=tf.keras.optimizers.RMSprop(
    learning_rate=param1['lr'],
    name="RMSprop"
), metrics=['accuracy'])

model1.summary()
```

et un autre modèle avec 32 + 64 + 128 + 256 couches

```
model2 = Sequential()

model2.add(Conv2D(32, (3, 3), activation=param2['activation'],
input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS)))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.25))

model2.add(Conv2D(64, (3, 3), activation=param2['activation']))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.25))

model2.add(Conv2D(128, (3, 3), activation=param2['activation']))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.25))

#On ajoute un layer
model2.add(Conv2D(256, (3, 3), activation=param2['activation']))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.25))

model2.add(Flatten())
model2.add(Dense(512, activation=param2['activation']))
model2.add(BatchNormalization())
model2.add(Dropout(0.5))
model2.add(Dense(2, activation='softmax')) # 2 because we have cat and
dog classes

model2.compile(loss=param2['loss'],
optimizer=tf.keras.optimizers.RMSprop(
    learning_rate=param2['lr'],
    name="RMSprop"
)
, metrics=['accuracy'])

model2.summary()
```

On ajoute pour chaque des batch normalization pour rendre le modèle plus stable et des drop out pour l'overfitting. La fonction d'activation de fin est softmax.

J'ai ensuite créé différents jeux de paramètre en faisant varier la fonction loss, l'optimizer, le learning rate, la fonction d'activation, le nombre d'epochs et le batch size.

Pour chaque entraînement, on utilise 70% du dataset pour le train, 15% du dataset pour la validation et 10% du dataset pour le test.

```
param1 = {
    'loss' : 'categorical_crossentropy',
    'optimizer' : 'rmsprop',
    'lr' : 0.001,
    'activation' : 'relu',
    'ratioTTV' : (0.75,0.15,0.10),
    'epoch' : 25,
    'batch_size': 20
}

param2 = {
    'loss' : 'categorical_crossentropy',
    'optimizer' : 'rmsprop',
    'lr' : 0.001,
    'activation' : 'relu',
    'ratioTTV' : (0.75,0.15,0.10),
    'epoch' : 25,
    'batch_size': 20
} # Pas de différence mais on ajoute un layer sur le model

param3 = {
    'loss' : 'categorical_crossentropy',
    'optimizer' : 'rmsprop',
    'lr' : 0.001,
    'activation' : 'relu',
    'ratioTTV' : (0.75,0.15,0.10),
    'epoch' : 50, # on teste avec plus d'époques
    'batch_size': 20
}


param4 = {
    'loss' : 'binary_crossentropy', # on change la fonction de loss
    'optimizer' : 'rmsprop',
    'lr' : 0.001,
    'activation' : 'relu',
    'ratioTTV' : (0.75,0.15,0.10),
    'epoch' : 50,
    'batch_size': 20
}

param5 = {
    'loss' : 'binary_crossentropy',
```

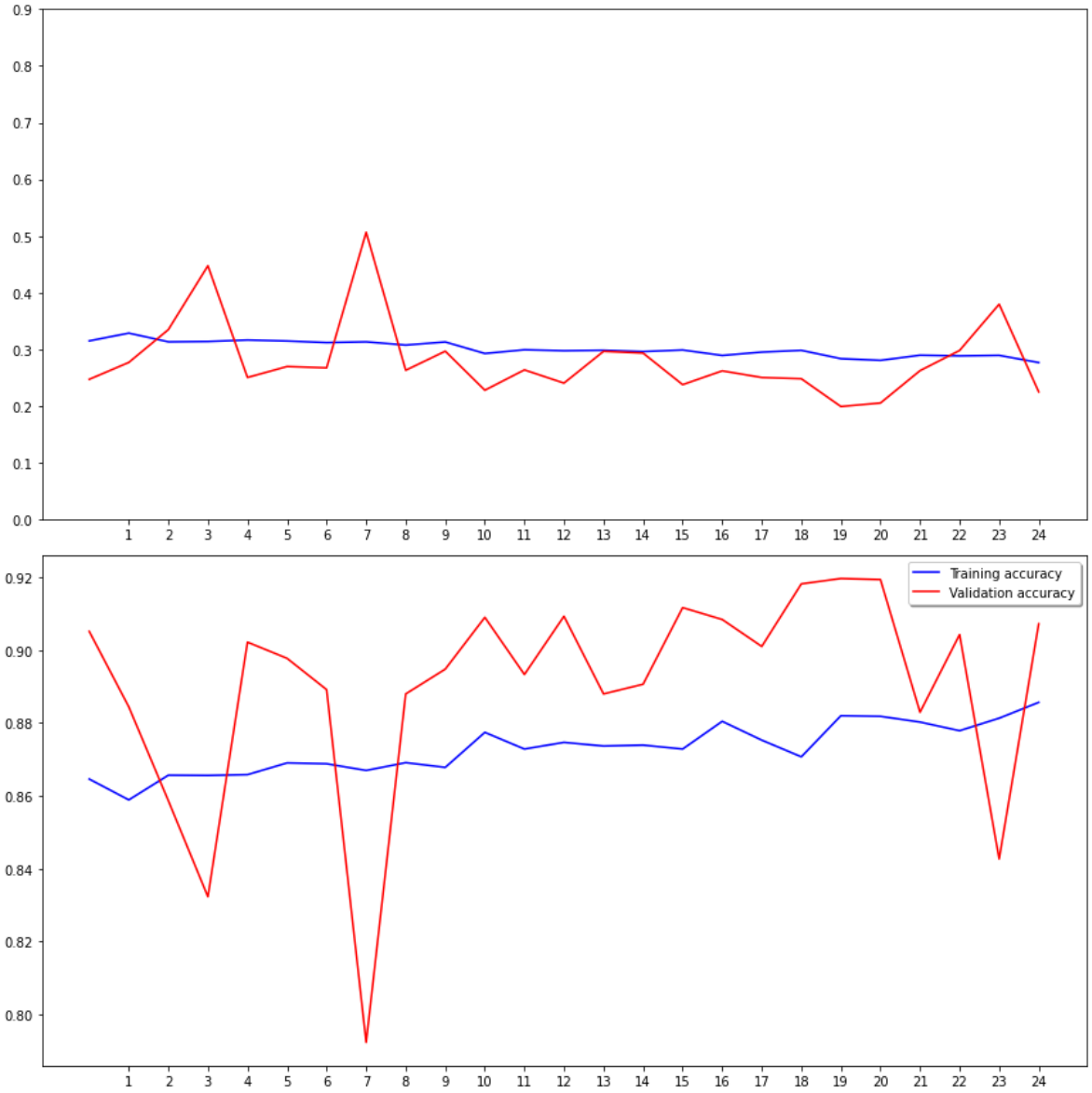
```
    'optimizer' : 'adam',  # on change l'optimizer
    'lr' : 0.001,
    'activation' : 'relu',
    'ratioTTV' : (0.75,0.15,0.10), # Ratio train,validation,test
    'epoch' : 50,
    'batch_size': 15
} #Meilleur résultat avec ces paramètres

param6 = {
    'loss' : 'binary_crossentropy',
    'optimizer' : 'adam',
    'lr' : 0.005, # on change le learning rate
    'activation' : 'relu',
    'ratioTTV' : (0.75,0.15,0.10), # Ratio train,validation,test
    'epoch' : 50,
    'batch_size': 15
}
```

J'ai appliqué le paramètre 1 au modèle 1 puis pour le modèle 2 j'ai appliqué les paramètres 2 à 6.

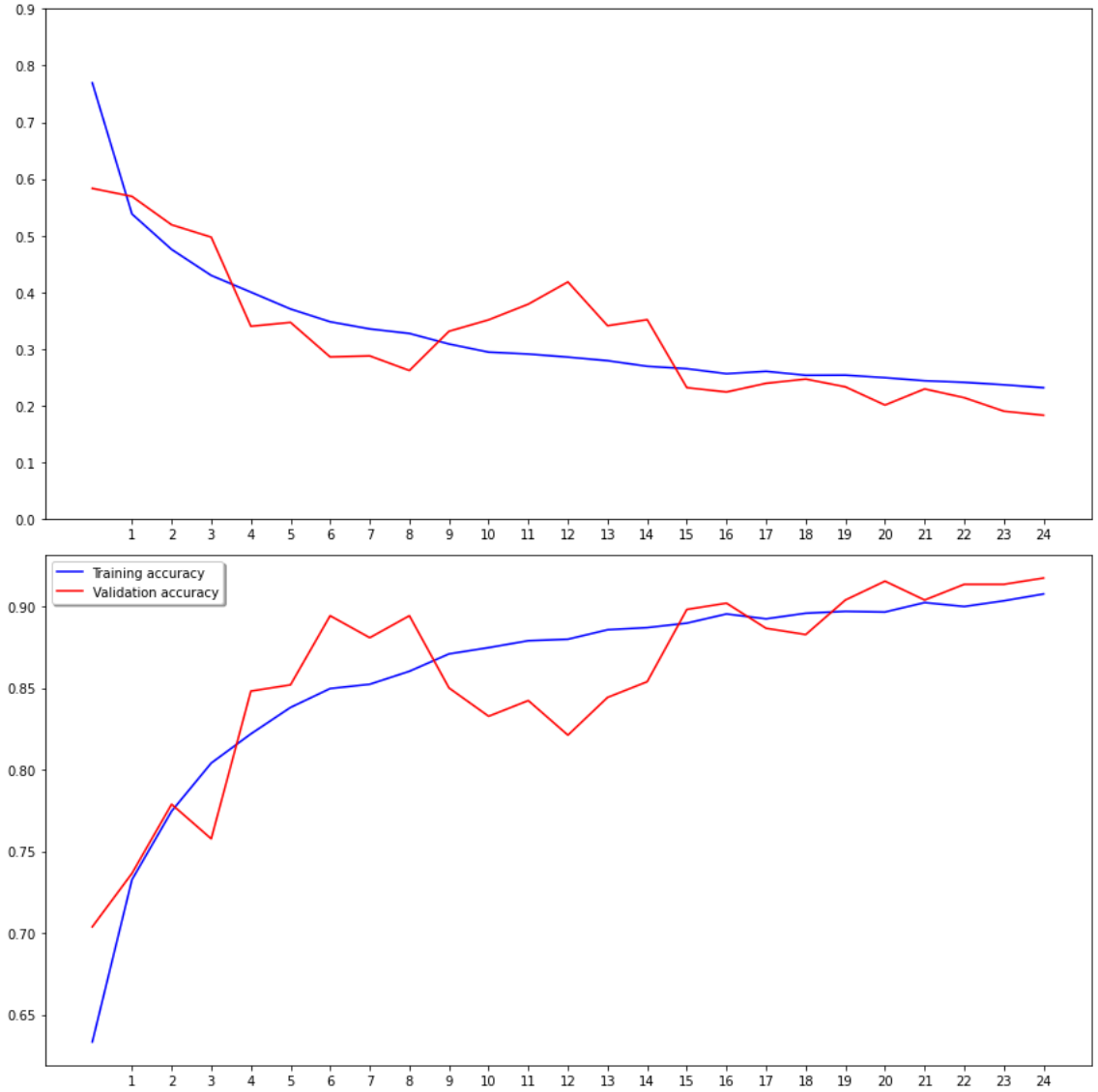
Pour chaque entraînement, on ajoute un callback qui va réduire le learning rate lorsque la val_accuracy stagne. Il y aussi un early stop qui va arrêter l'entraînement lorsque val_accuracy stagne au bout de 10 epoch.

```
earlystop = EarlyStopping(patience=10)

learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=10,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.001)

callbacks = [earlystop, learning_rate_reduction]
```

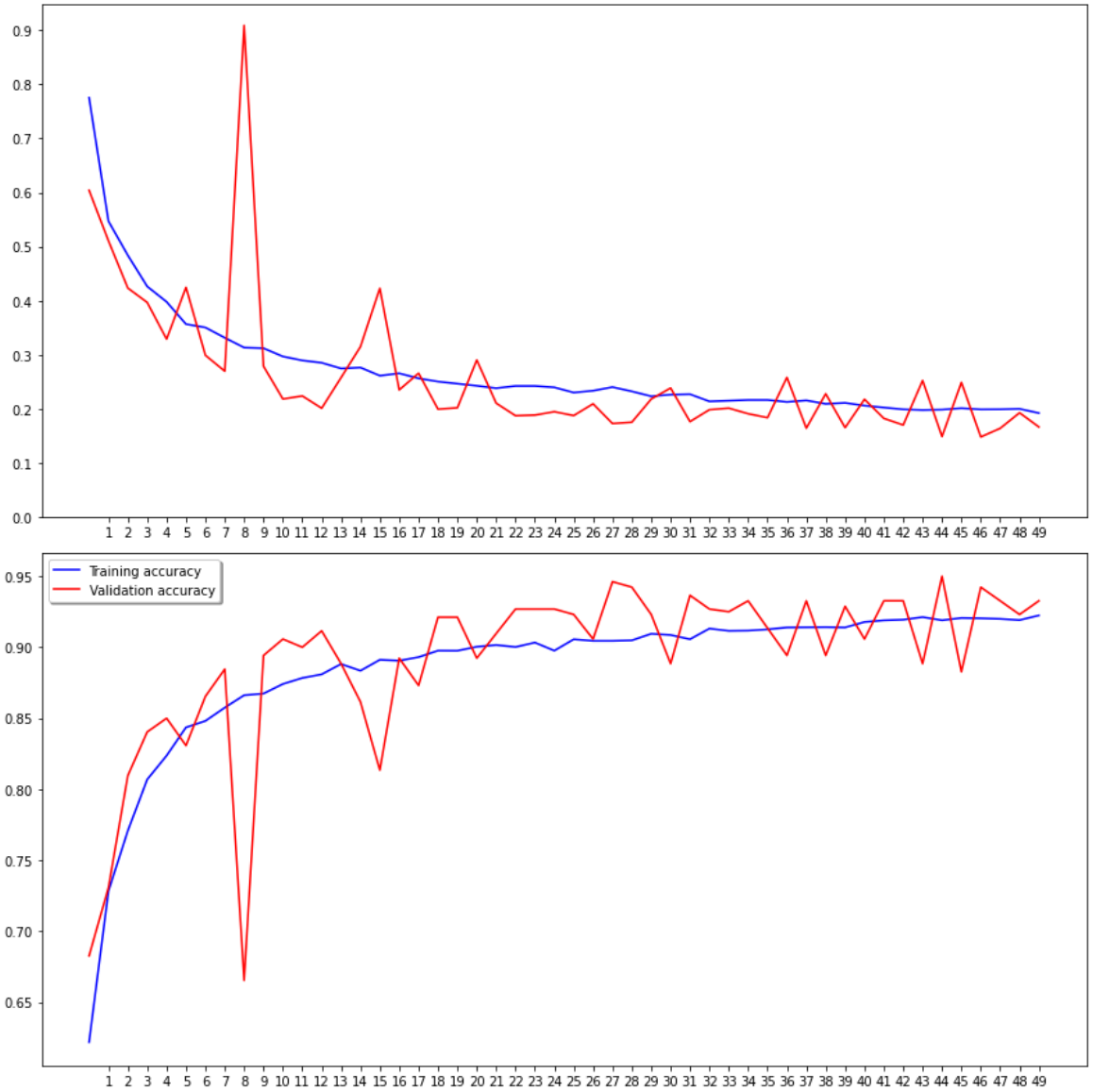
Résultats d'entrainements



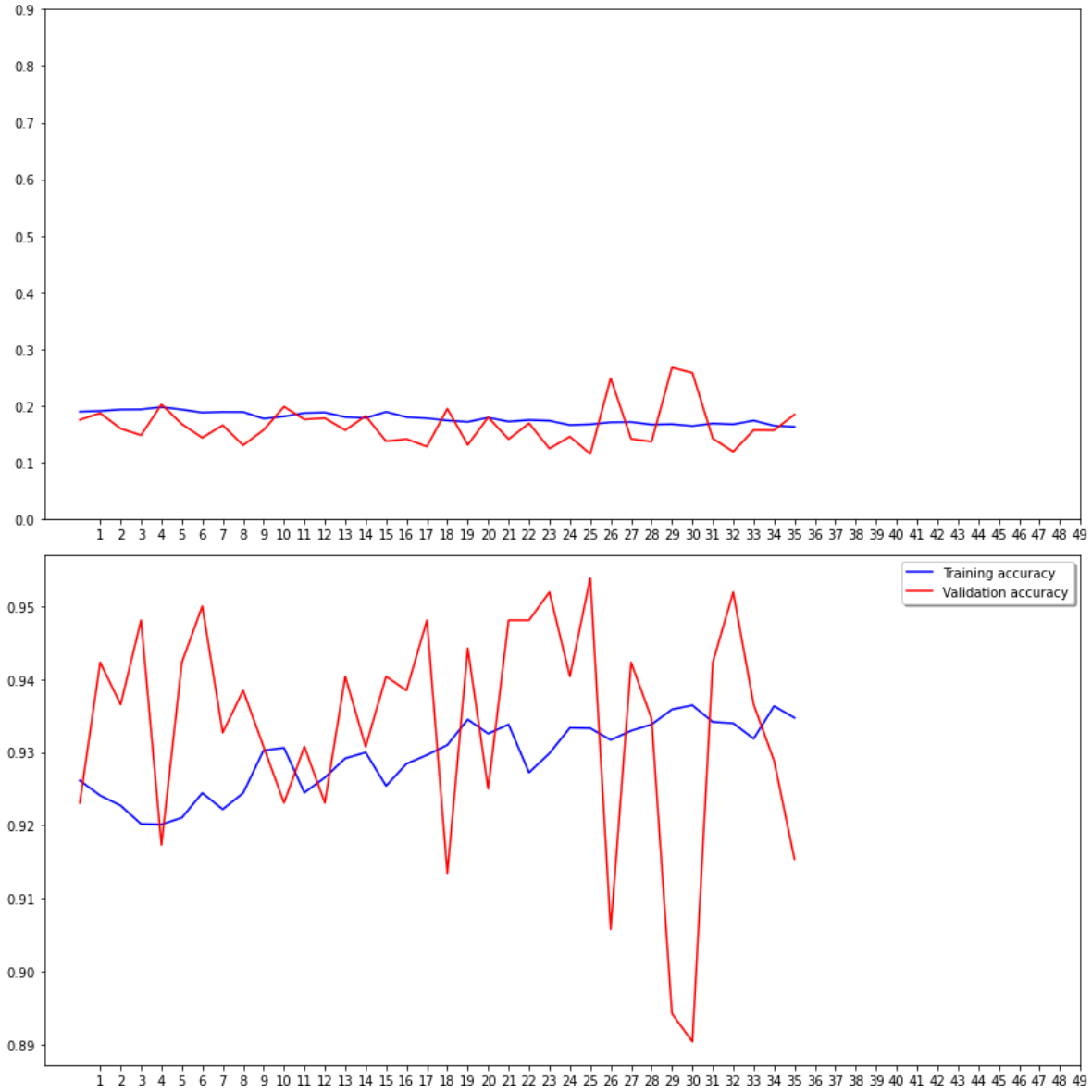
Paramètre 1 : loss: 0.2769 - accuracy: 0.8857 - val_loss: 0.2252 - val_accuracy: 0.9073 - lr: 0.0010



Paramètre 2 : loss: 0.2317 - accuracy: 0.9075 - val_loss: 0.1832 - val_accuracy: 0.9173 - lr: 0.0010

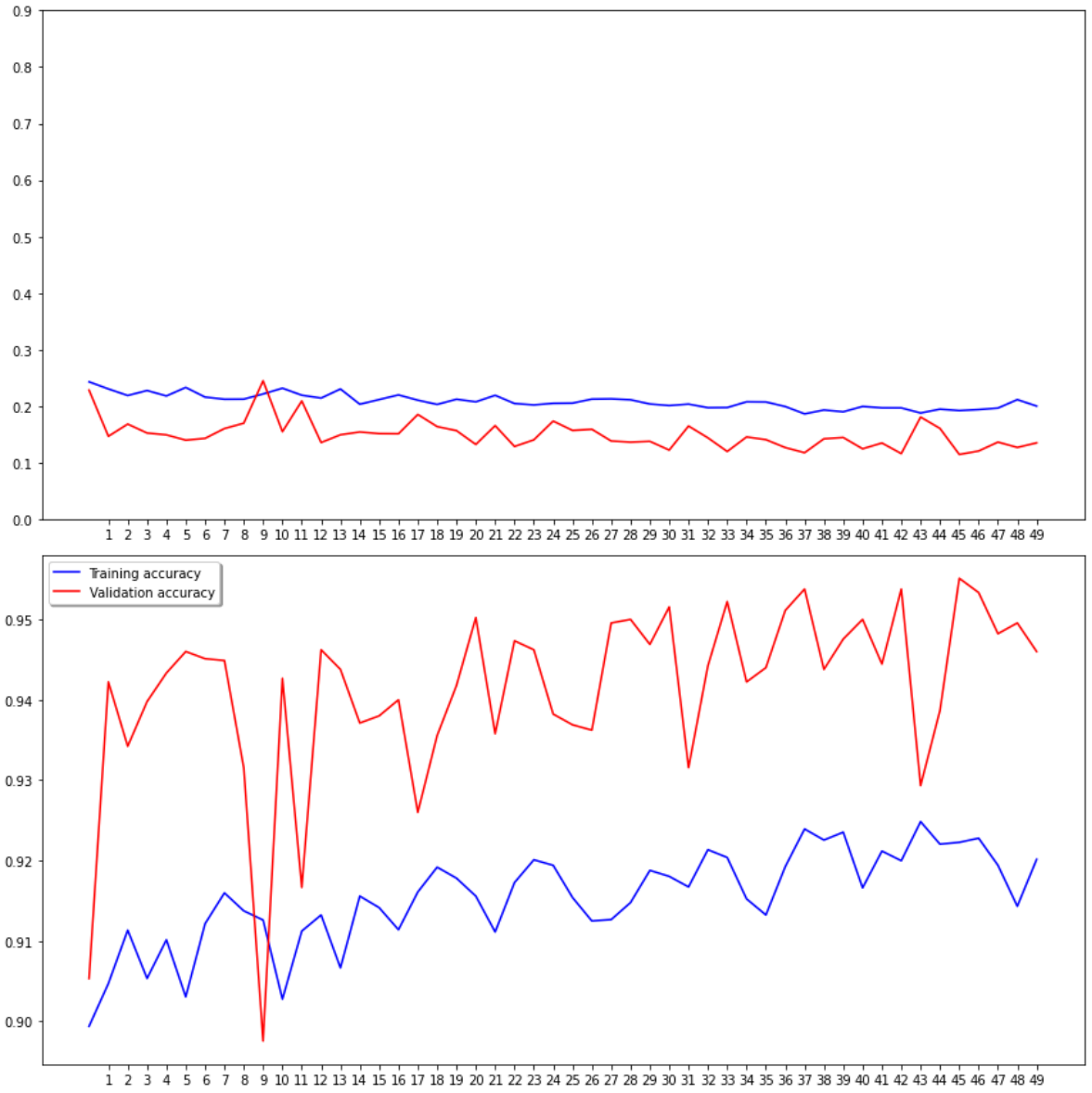


Paramètre 3 : loss: 0.1924 - accuracy: 0.9224 - val_loss: 0.1667 - val_accuracy: 0.9327 - lr: 0.0010

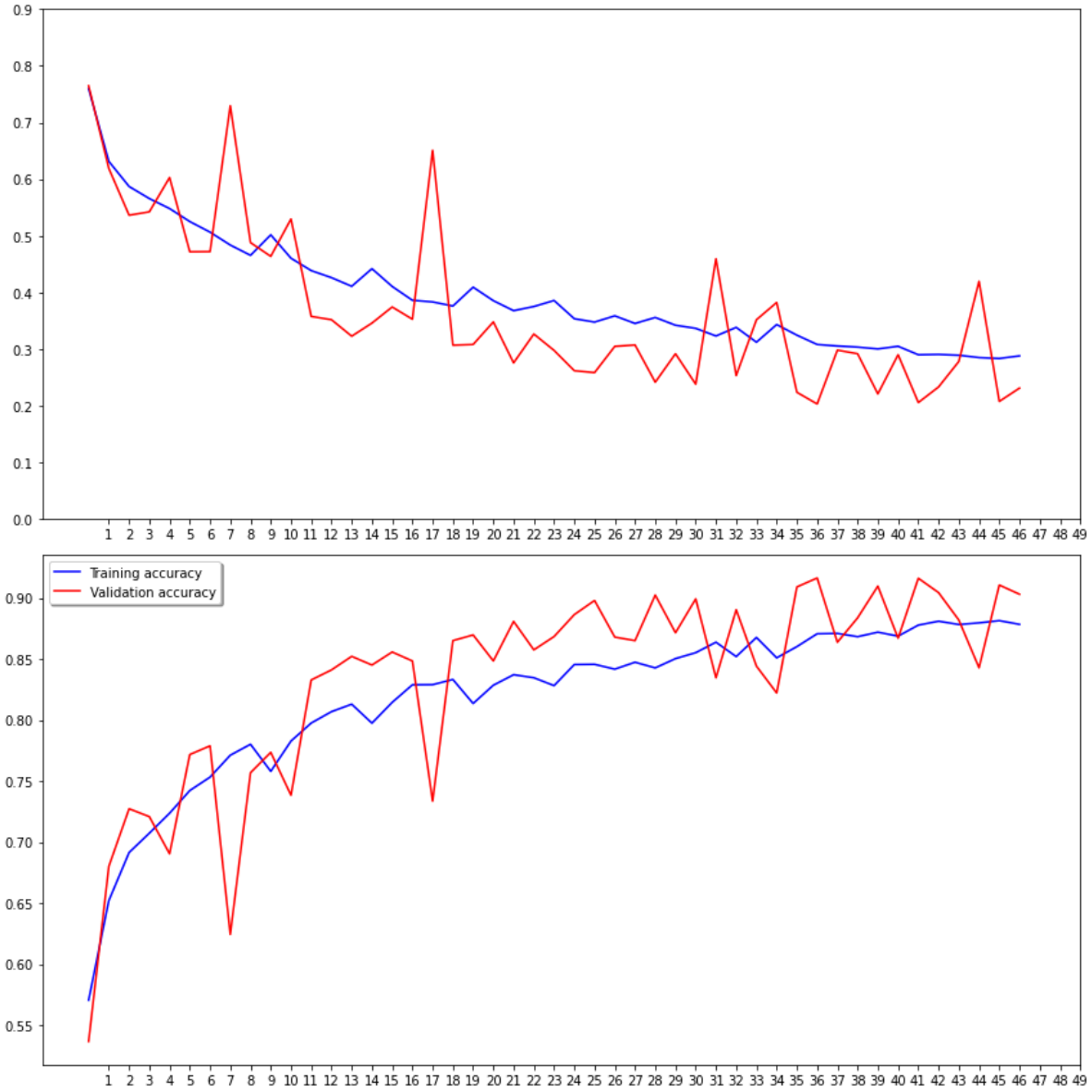


Paramètre 4 : loss: 0.1631 - accuracy: 0.9347 - val_loss: 0.1847 - val_accuracy: 0.9154 - lr: 0.0010

On a un early stop à partir de la 35 ème époque.



Paramètre 5 : loss: 0.2005 - accuracy: 0.9202 - val_loss: 0.1355 - val_accuracy: 0.9460 - lr: 0.0010

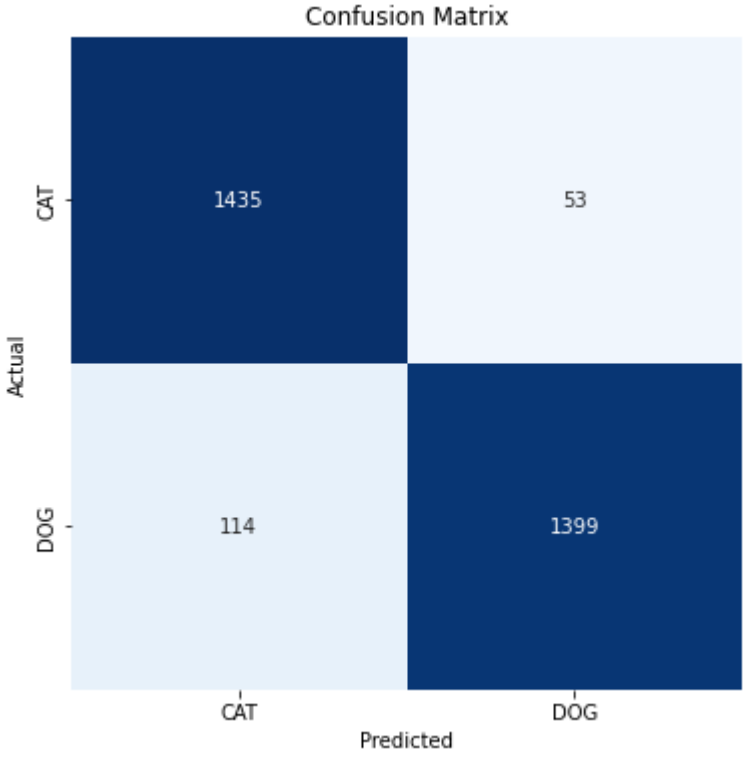


Paramètre 6 : loss: loss: 0.2881 - accuracy: 0.8785 - val_loss: 0.2312 - val_accuracy: 0.9031 - lr: 0.0050

Pour la majorité des modèles, la validation semble peu stable. Il aurait été plus judicieux de faire un peu plus varié le learning rate.

Meilleur résultat de test

C'est le paramètre 5 qui a obtenu le meilleur résultat avec une accuracy de 0.9443518.


Confusion Matrix
1435
53
114
1399
CAT
DOG
Actual
CAT
DOG
Predicted