ex1_functional_api

May 5, 2024

1 Exercise 1

[ ]:
```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten,␣
↪Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import numpy as np
from sklearn import metrics as me
from tensorflow.keras.layers import concatenate
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import load_model
```

[ ]:
```
# Load data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Normalize pixel values to be between 0 and 1
X_train, X_test = X_train / 255.0, X_test / 255.0

# Convert class vectors to binary class matrices (for use in␣
↪categorical_crossentropy)
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

[ ]:
```
# Define the checkpoint directory and file
checkpoint_filepath = '/tmp/checkpoint'
checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    monitor='val_accuracy',
    verbose=1,
    save_best_only=True,
    mode='max'
)
```

1.1 Sequential strategy using the functional API

[ ]:
```
# Input shape is the shape of CIFAR-10 images
input_shape = (32, 32, 3)

# Define the input tensor
inputs = Input(shape=input_shape)

# Define the layers
x = Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2))(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2))(x)
x = Flatten()(x)
x = Dropout(0.3)(x)
x = Dense(300, activation='relu')(x)
x = Dropout(0.3)(x)
outputs = Dense(10, activation='softmax')(x)

# Create the model
model1 = Model(inputs=inputs, outputs=outputs)

# Model summary
model1.summary()
```

```
Model: "model"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 conv2d (Conv2D)             (None, 32, 32, 32)        896

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d (MaxPooling2  (None, 16, 16, 32)        0
 D)

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 64)          0
```

```
 g2D)

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 128)         0
 g2D)

 flatten (Flatten)           (None, 2048)              0

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 300)               614700

 dropout_1 (Dropout)         (None, 300)               0

 dense_1 (Dense)             (None, 10)                3010

=================================================================
Total params: 904718 (3.45 MB)
Trainable params: 904718 (3.45 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

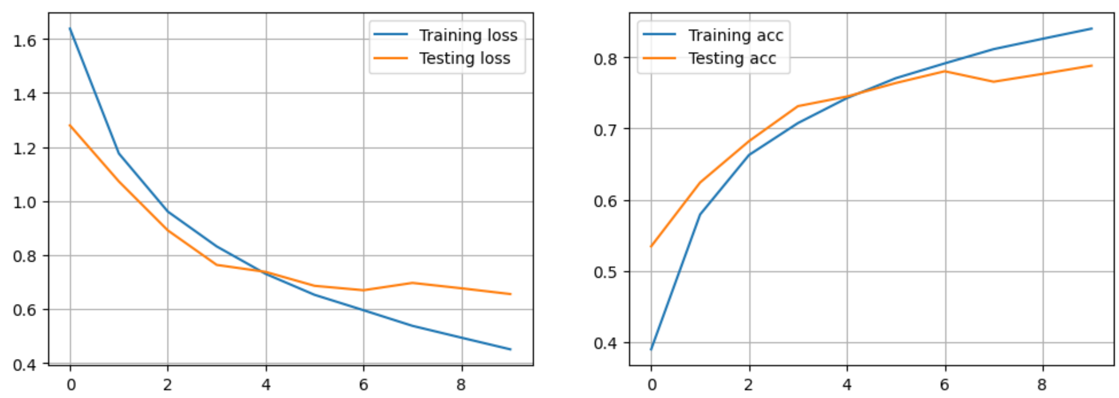
[ ]:
```
model1.compile(optimizer=Adam(), loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Train the model
log_model1 = model1.fit(X_train, y_train, epochs=10, validation_split=0.2,␣
 ↪batch_size=64, callbacks=[checkpoint_callback])
```

```
Epoch 1/10
625/625 [==============================] - ETA: 0s - loss: 1.6383 - accuracy:
0.3900
Epoch 1: val_accuracy improved from -inf to 0.53460, saving model to
/tmp/checkpoint
625/625 [==============================] - 13s 12ms/step - loss: 1.6383 -
accuracy: 0.3900 - val_loss: 1.2801 - val_accuracy: 0.5346
Epoch 2/10
622/625 [============================>.] - ETA: 0s - loss: 1.1769 - accuracy:
0.5791
Epoch 2: val_accuracy improved from 0.53460 to 0.62420, saving model to
/tmp/checkpoint
625/625 [==============================] - 7s 11ms/step - loss: 1.1761 -
accuracy: 0.5793 - val_loss: 1.0727 - val_accuracy: 0.6242
Epoch 3/10
621/625 [============================>.] - ETA: 0s - loss: 0.9620 - accuracy:
```

```
0.6625
Epoch 3: val_accuracy improved from 0.62420 to 0.68230, saving model to
/tmp/checkpoint
625/625 [==============================] - 6s 10ms/step - loss: 0.9612 -
accuracy: 0.6629 - val_loss: 0.8915 - val_accuracy: 0.6823
Epoch 4/10
621/625 [============================>.] - ETA: 0s - loss: 0.8327 - accuracy:
0.7074
Epoch 4: val_accuracy improved from 0.68230 to 0.73140, saving model to
/tmp/checkpoint
625/625 [==============================] - 7s 11ms/step - loss: 0.8321 -
accuracy: 0.7077 - val_loss: 0.7636 - val_accuracy: 0.7314
Epoch 5/10
619/625 [============================>.] - ETA: 0s - loss: 0.7312 - accuracy:
0.7428
Epoch 5: val_accuracy improved from 0.73140 to 0.74490, saving model to
/tmp/checkpoint
625/625 [==============================] - 7s 10ms/step - loss: 0.7305 -
accuracy: 0.7427 - val_loss: 0.7378 - val_accuracy: 0.7449
Epoch 6/10
621/625 [============================>.] - ETA: 0s - loss: 0.6529 - accuracy:
0.7705
Epoch 6: val_accuracy improved from 0.74490 to 0.76390, saving model to
/tmp/checkpoint
625/625 [==============================] - 7s 11ms/step - loss: 0.6529 -
accuracy: 0.7707 - val_loss: 0.6862 - val_accuracy: 0.7639
Epoch 7/10
620/625 [============================>.] - ETA: 0s - loss: 0.5963 - accuracy:
0.7913
Epoch 7: val_accuracy improved from 0.76390 to 0.78040, saving model to
/tmp/checkpoint
625/625 [==============================] - 6s 10ms/step - loss: 0.5959 -
accuracy: 0.7915 - val_loss: 0.6699 - val_accuracy: 0.7804
Epoch 8/10
624/625 [============================>.] - ETA: 0s - loss: 0.5382 - accuracy:
0.8113
Epoch 8: val_accuracy did not improve from 0.78040
625/625 [==============================] - 6s 9ms/step - loss: 0.5380 -
accuracy: 0.8115 - val_loss: 0.6970 - val_accuracy: 0.7657
Epoch 9/10
624/625 [============================>.] - ETA: 0s - loss: 0.4942 - accuracy:
0.8259
Epoch 9: val_accuracy did not improve from 0.78040
625/625 [==============================] - 5s 8ms/step - loss: 0.4943 -
accuracy: 0.8260 - val_loss: 0.6771 - val_accuracy: 0.7767
Epoch 10/10
623/625 [============================>.] - ETA: 0s - loss: 0.4511 - accuracy:
0.8404
```

```
Epoch 10: val_accuracy improved from 0.78040 to 0.78820, saving model to
/tmp/checkpoint
625/625 [==============================] - 7s 11ms/step - loss: 0.4511 -
accuracy: 0.8402 - val_loss: 0.6558 - val_accuracy: 0.7882
```

[ ]:
```
f = plt.figure(figsize=(12,4))
ax1 = f.add_subplot(121)
ax2 = f.add_subplot(122)
ax1.plot(log_model1.history['loss'], label='Training loss')
ax1.plot(log_model1.history['val_loss'], label='Testing loss')
ax1.legend()
ax1.grid()
ax2.plot(log_model1.history['accuracy'], label='Training acc')
ax2.plot(log_model1.history['val_accuracy'], label='Testing acc')
ax2.legend()
ax2.grid()
```



[ ]:
```
loss_test, metric_test = model1.evaluate(X_test, y_test, verbose=0)
print('model test loss:', loss_test)
print('model test accuracy:', metric_test)
```

```
model test loss: 0.6863976716995239
model test accuracy: 0.7795000076293945
```

1.2 Multiple paths strategy

[ ]:
```
# Input layer
inputs = Input(shape=(32, 32, 3))

# Path 1: Larger filters
path1 = Conv2D(32, (5, 5), activation='relu', padding='same')(inputs)
path1 = MaxPooling2D((2, 2))(path1)
path1 = Conv2D(64, (3, 3), activation='relu', padding='same')(path1)
```

```
path1 = MaxPooling2D((2, 2))(path1)

# Path 2: Smaller filters
path2 = Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
path2 = Conv2D(32, (3, 3), activation='relu', padding='same')(path2)
path2 = MaxPooling2D((2, 2))(path2)
path2 = Conv2D(64, (3, 3), activation='relu', padding='same')(path2)
path2 = Conv2D(64, (3, 3), activation='relu', padding='same')(path2)
path2 = MaxPooling2D((2, 2))(path2)

# Concatenate both paths
combined = concatenate([path1, path2])

# Follow-up layers
x = Flatten()(combined)
x = Dropout(0.3)(x)
x = Dense(300, activation='relu')(x)
x = Dropout(0.3)(x)
outputs = Dense(10, activation='softmax')(x)

# Create the model
model2 = Model(inputs=inputs, outputs=outputs)
model2.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
model2.summary()
```

```
Model: "model_1"
________________________________________________________________________________
__________________
 Layer (type)                Output Shape                 Param #   Connected to
================================================================================
==================
 input_2 (InputLayer)        [(None, 32, 32, 3)]          0         []

 conv2d_8 (Conv2D)           (None, 32, 32, 32)           896
['input_2[0][0]']

 conv2d_9 (Conv2D)           (None, 32, 32, 32)           9248
['conv2d_8[0][0]']

 conv2d_6 (Conv2D)           (None, 32, 32, 32)           2432
['input_2[0][0]']

 max_pooling2d_5 (MaxPoolin  (None, 16, 16, 32)           0
['conv2d_9[0][0]']
 g2D)

 max_pooling2d_3 (MaxPoolin  (None, 16, 16, 32)           0
```

```
['conv2d_6[0][0]']
 g2D)

 conv2d_10 (Conv2D)          (None, 16, 16, 64)           18496
['max_pooling2d_5[0][0]']

 conv2d_7 (Conv2D)           (None, 16, 16, 64)           18496
['max_pooling2d_3[0][0]']

 conv2d_11 (Conv2D)          (None, 16, 16, 64)           36928
['conv2d_10[0][0]']

 max_pooling2d_4 (MaxPoolin  (None, 8, 8, 64)             0
['conv2d_7[0][0]']
 g2D)

 max_pooling2d_6 (MaxPoolin  (None, 8, 8, 64)             0
['conv2d_11[0][0]']
 g2D)

 concatenate (Concatenate)   (None, 8, 8, 128)            0
['max_pooling2d_4[0][0]',
'max_pooling2d_6[0][0]']

 flatten_1 (Flatten)         (None, 8192)                 0
['concatenate[0][0]']

 dropout_2 (Dropout)         (None, 8192)                 0
['flatten_1[0][0]']

 dense_2 (Dense)             (None, 300)                  2457900
['dropout_2[0][0]']

 dropout_3 (Dropout)         (None, 300)                  0
['dense_2[0][0]']

 dense_3 (Dense)             (None, 10)                   3010
['dropout_3[0][0]']

================================================================================
==================
Total params: 2547406 (9.72 MB)
Trainable params: 2547406 (9.72 MB)
Non-trainable params: 0 (0.00 Byte)
--------------------------------------------------------------------------------
------------------
```

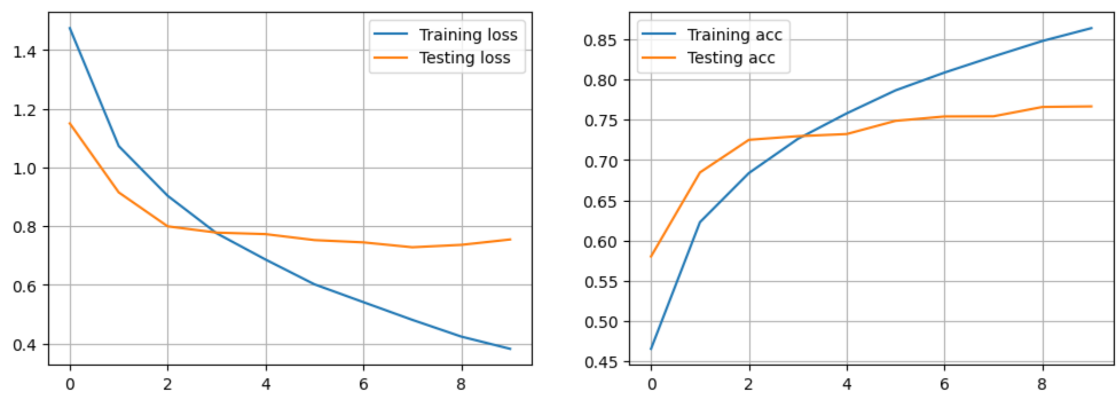
[ ]:

```
log_model2 = model2.fit(X_train, y_train, epochs=10, validation_split=0.2,␣
↪batch_size=64, callbacks=[checkpoint_callback])
```

```
Epoch 1/10
625/625 [==============================] - ETA: 0s - loss: 1.4745 - accuracy:
0.4656
Epoch 1: val_accuracy did not improve from 0.78820
625/625 [==============================] - 10s 11ms/step - loss: 1.4745 -
accuracy: 0.4656 - val_loss: 1.1500 - val_accuracy: 0.5803
Epoch 2/10
622/625 [============================>.] - ETA: 0s - loss: 1.0734 - accuracy:
0.6228
Epoch 2: val_accuracy did not improve from 0.78820
625/625 [==============================] - 5s 9ms/step - loss: 1.0728 -
accuracy: 0.6229 - val_loss: 0.9149 - val_accuracy: 0.6846
Epoch 3/10
622/625 [============================>.] - ETA: 0s - loss: 0.9042 - accuracy:
0.6836
Epoch 3: val_accuracy did not improve from 0.78820
625/625 [==============================] - 6s 9ms/step - loss: 0.9034 -
accuracy: 0.6838 - val_loss: 0.7993 - val_accuracy: 0.7251
Epoch 4/10
623/625 [============================>.] - ETA: 0s - loss: 0.7754 - accuracy:
0.7263
Epoch 4: val_accuracy did not improve from 0.78820
625/625 [==============================] - 6s 9ms/step - loss: 0.7758 -
accuracy: 0.7262 - val_loss: 0.7780 - val_accuracy: 0.7296
Epoch 5/10
621/625 [============================>.] - ETA: 0s - loss: 0.6864 - accuracy:
0.7580
Epoch 5: val_accuracy did not improve from 0.78820
625/625 [==============================] - 6s 10ms/step - loss: 0.6862 -
accuracy: 0.7580 - val_loss: 0.7728 - val_accuracy: 0.7323
Epoch 6/10
622/625 [============================>.] - ETA: 0s - loss: 0.6010 - accuracy:
0.7869
Epoch 6: val_accuracy did not improve from 0.78820
625/625 [==============================] - 5s 9ms/step - loss: 0.6018 -
accuracy: 0.7866 - val_loss: 0.7524 - val_accuracy: 0.7487
Epoch 7/10
622/625 [============================>.] - ETA: 0s - loss: 0.5404 - accuracy:
0.8087
Epoch 7: val_accuracy did not improve from 0.78820
625/625 [==============================] - 6s 10ms/step - loss: 0.5411 -
accuracy: 0.8087 - val_loss: 0.7447 - val_accuracy: 0.7541
Epoch 8/10
620/625 [============================>.] - ETA: 0s - loss: 0.4809 - accuracy:
0.8285
```

```
Epoch 8: val_accuracy did not improve from 0.78820
625/625 [==============================] - 6s 9ms/step - loss: 0.4808 -
accuracy: 0.8286 - val_loss: 0.7280 - val_accuracy: 0.7543
Epoch 9/10
623/625 [============================>.] - ETA: 0s - loss: 0.4234 - accuracy:
0.8480
Epoch 9: val_accuracy did not improve from 0.78820
625/625 [==============================] - 6s 10ms/step - loss: 0.4238 -
accuracy: 0.8478 - val_loss: 0.7362 - val_accuracy: 0.7659
Epoch 10/10
623/625 [============================>.] - ETA: 0s - loss: 0.3822 - accuracy:
0.8637
Epoch 10: val_accuracy did not improve from 0.78820
625/625 [==============================] - 6s 10ms/step - loss: 0.3820 -
accuracy: 0.8637 - val_loss: 0.7545 - val_accuracy: 0.7666
```

[ ]:
```
f = plt.figure(figsize=(12,4))
ax1 = f.add_subplot(121)
ax2 = f.add_subplot(122)
ax1.plot(log_model2.history['loss'], label='Training loss')
ax1.plot(log_model2.history['val_loss'], label='Testing loss')
ax1.legend()
ax1.grid()
ax2.plot(log_model2.history['accuracy'], label='Training acc')
ax2.plot(log_model2.history['val_accuracy'], label='Testing acc')
ax2.legend()
ax2.grid()
```



[ ]:
```
test_loss, test_acc = model2.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}, Test loss: {test_loss}')
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.7801 -
accuracy: 0.7501
Test accuracy: 0.7501000165939331, Test loss: 0.7801382541656494
```

1.3 Multiple features strategy

[ ]:
```python
# Input layer
visible = Input(shape=(32, 32, 3))

# First feature extractor
conv1 = Conv2D(32, kernel_size=3, activation='relu')(visible)
drop1 = Dropout(0.2)(conv1)
pool1 = MaxPooling2D(pool_size=(2, 2))(drop1)
flat1 = Flatten()(pool1)

# Second feature extractor
conv2 = Conv2D(32, kernel_size=3, activation='relu')(pool1)
drop2 = Dropout(0.2)(conv2)
pool2 = MaxPooling2D(pool_size=(2, 2))(drop2)
flat2 = Flatten()(pool2)

# Third feature extractor
conv3 = Conv2D(32, kernel_size=3, activation='relu')(pool2)
drop3 = Dropout(0.2)(conv3)
pool3 = MaxPooling2D(pool_size=(2, 2))(drop3)
flat3 = Flatten()(pool3)

# Merge feature extractors
merge = concatenate([flat1, flat2, flat3])

# Interpretation layer
hidden1 = Dense(100, activation='relu')(merge)

# Prediction output
output = Dense(10, activation='softmax')(hidden1)

# Create the model
model3 = Model(inputs=visible, outputs=output)

# Compile the model
model3.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

# Summarize layers
print(model3.summary())
```

```
Model: "model_2"

__________________________________________________________________________________
________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================
================
```

```
 input_3 (InputLayer)        [(None, 32, 32, 3)]          0           []

 conv2d_12 (Conv2D)          (None, 30, 30, 32)           896
['input_3[0][0]']

 dropout_4 (Dropout)         (None, 30, 30, 32)           0
['conv2d_12[0][0]']

 max_pooling2d_7 (MaxPoolin  (None, 15, 15, 32)           0
['dropout_4[0][0]']
 g2D)

 conv2d_13 (Conv2D)          (None, 13, 13, 32)           9248
['max_pooling2d_7[0][0]']

 dropout_5 (Dropout)         (None, 13, 13, 32)           0
['conv2d_13[0][0]']

 max_pooling2d_8 (MaxPoolin  (None, 6, 6, 32)             0
['dropout_5[0][0]']
 g2D)

 conv2d_14 (Conv2D)          (None, 4, 4, 32)             9248
['max_pooling2d_8[0][0]']

 dropout_6 (Dropout)         (None, 4, 4, 32)             0
['conv2d_14[0][0]']

 max_pooling2d_9 (MaxPoolin  (None, 2, 2, 32)             0
['dropout_6[0][0]']
 g2D)

 flatten_2 (Flatten)         (None, 7200)                 0
['max_pooling2d_7[0][0]']

 flatten_3 (Flatten)         (None, 1152)                 0
['max_pooling2d_8[0][0]']

 flatten_4 (Flatten)         (None, 128)                  0
['max_pooling2d_9[0][0]']

 concatenate_1 (Concatenate  (None, 8480)                 0
['flatten_2[0][0]',
 )
'flatten_3[0][0]',
'flatten_4[0][0]']

 dense_4 (Dense)             (None, 100)                  848100
```

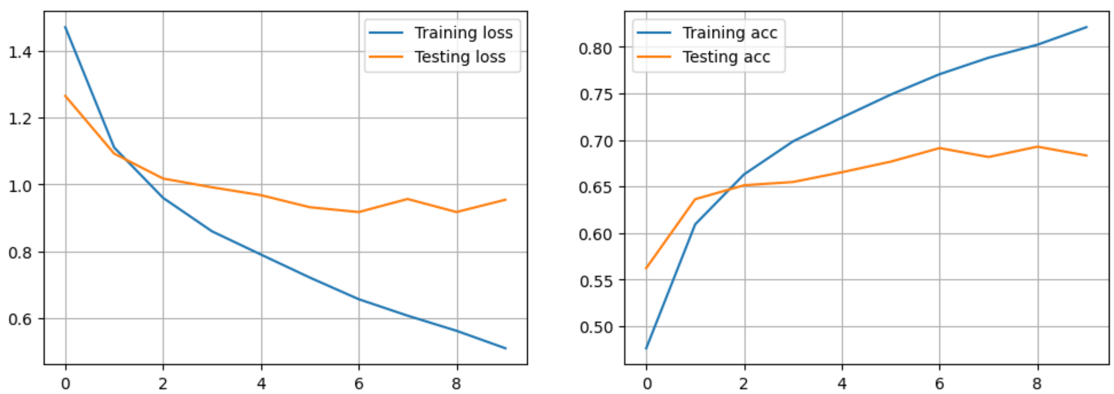
```
['concatenate_1[0][0]']

 dense_5 (Dense)             (None, 10)                   1010
['dense_4[0][0]']

================================================================================
==================
Total params: 868502 (3.31 MB)
Trainable params: 868502 (3.31 MB)
Non-trainable params: 0 (0.00 Byte)
--------------------------------------------------------------------------------
------------------
None
```

[ ]:
```
log_model3 = model3.fit(X_train, y_train, epochs=10, validation_split=0.2,␣
↪batch_size=64, callbacks=[checkpoint_callback])
```

```
Epoch 1/10
619/625 [============================>.] - ETA: 0s - loss: 1.4714 - accuracy:
0.4754
Epoch 1: val_accuracy did not improve from 0.78820
625/625 [==============================] - 7s 7ms/step - loss: 1.4697 -
accuracy: 0.4760 - val_loss: 1.2645 - val_accuracy: 0.5622
Epoch 2/10
618/625 [============================>.] - ETA: 0s - loss: 1.1117 - accuracy:
0.6086
Epoch 2: val_accuracy did not improve from 0.78820
625/625 [==============================] - 4s 6ms/step - loss: 1.1102 -
accuracy: 0.6093 - val_loss: 1.0921 - val_accuracy: 0.6363
Epoch 3/10
623/625 [============================>.] - ETA: 0s - loss: 0.9600 - accuracy:
0.6628
Epoch 3: val_accuracy did not improve from 0.78820
625/625 [==============================] - 4s 6ms/step - loss: 0.9599 -
accuracy: 0.6630 - val_loss: 1.0176 - val_accuracy: 0.6513
Epoch 4/10
614/625 [============================>.] - ETA: 0s - loss: 0.8596 - accuracy:
0.6985
Epoch 4: val_accuracy did not improve from 0.78820
625/625 [==============================] - 4s 7ms/step - loss: 0.8600 -
accuracy: 0.6985 - val_loss: 0.9915 - val_accuracy: 0.6548
Epoch 5/10
616/625 [============================>.] - ETA: 0s - loss: 0.7911 - accuracy:
0.7237
Epoch 5: val_accuracy did not improve from 0.78820
625/625 [==============================] - 3s 5ms/step - loss: 0.7906 -
accuracy: 0.7241 - val_loss: 0.9680 - val_accuracy: 0.6653
Epoch 6/10
```

```
621/625 [============================>.] - ETA: 0s - loss: 0.7216 - accuracy:
0.7488
Epoch 6: val_accuracy did not improve from 0.78820
625/625 [==============================] - 4s 7ms/step - loss: 0.7219 -
accuracy: 0.7487 - val_loss: 0.9319 - val_accuracy: 0.6767
Epoch 7/10
625/625 [==============================] - ETA: 0s - loss: 0.6573 - accuracy:
0.7707
Epoch 7: val_accuracy did not improve from 0.78820
625/625 [==============================] - 4s 6ms/step - loss: 0.6573 -
accuracy: 0.7707 - val_loss: 0.9175 - val_accuracy: 0.6913
Epoch 8/10
620/625 [============================>.] - ETA: 0s - loss: 0.6084 - accuracy:
0.7881
Epoch 8: val_accuracy did not improve from 0.78820
625/625 [==============================] - 4s 6ms/step - loss: 0.6080 -
accuracy: 0.7884 - val_loss: 0.9565 - val_accuracy: 0.6817
Epoch 9/10
623/625 [============================>.] - ETA: 0s - loss: 0.5628 - accuracy:
0.8026
Epoch 9: val_accuracy did not improve from 0.78820
625/625 [==============================] - 4s 6ms/step - loss: 0.5629 -
accuracy: 0.8024 - val_loss: 0.9176 - val_accuracy: 0.6928
Epoch 10/10
620/625 [============================>.] - ETA: 0s - loss: 0.5108 - accuracy:
0.8211
Epoch 10: val_accuracy did not improve from 0.78820
625/625 [==============================] - 4s 6ms/step - loss: 0.5106 -
accuracy: 0.8212 - val_loss: 0.9543 - val_accuracy: 0.6833
```

[ ]:
```
f = plt.figure(figsize=(12,4))
ax1 = f.add_subplot(121)
ax2 = f.add_subplot(122)
ax1.plot(log_model3.history['loss'], label='Training loss')
ax1.plot(log_model3.history['val_loss'], label='Testing loss')
ax1.legend()
ax1.grid()
ax2.plot(log_model3.history['accuracy'], label='Training acc')
ax2.plot(log_model3.history['val_accuracy'], label='Testing acc')
ax2.legend()
ax2.grid()
```



[ ]:
```python
test_loss, test_acc = model3.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}, Test loss: {test_loss}')
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.9738 -
accuracy: 0.6717
Test accuracy: 0.6717000007629395, Test loss: 0.9737797975540161
```

1.4 Summary of results

The same hyperparameters were used for each training instance: - epochs: 10 - batch size: 64 - learning rate: 0.001

Model	Architecture	Callback	Acc. train %	Acc. test %
1	Conv2D(D=32, w=h=3, S=1, P=same) -> Conv2D(D=32, w=h=3, S=1, P=same) -> MaxPooling2D -> Conv2D(D=64, w=h=3, S=1, P=same) -> Conv2D(D=64, w=h=3, S=1, P=same) -> MaxPooling2D -> Conv2D(D=128, w=h=3, S=1, P=same) -> Conv2D(D=128, w=h=3, S=1, P=same) -> MaxPooling2D -> Flatten -> Dropout(0.3) -> Dense(300) -> Dropout(0.3) -> Dense(10)	yes	84.02%	67.95%
2	Path1: Conv2D(D=32, w=h=5, S=1, P=same) -> MaxPooling2D -> Conv2D(D=64, w=h=3, S=1, P=same) -> MaxPooling2D; Path2: Conv2D(D=32, w=h=3, S=1, P=same) x2 -> MaxPooling2D -> Conv2D(D=64, w=h=3, S=1, P=same) x2 -> MaxPooling2D -> Concatenate -> Flatten -> Dropout(0.3) -> Dense(300) -> Dropout(0.3) -> Dense(10)	yes	86.37%	75.01%
3	3 paths each with: Conv2D(D=32, w=h=3, S=1, P=same) -> Dropout(0.2) -> MaxPooling2D, then merged -> Flatten -> Dense(100) -> Dense(10)	yes	82.12%	67.17%

The sequential strategy performed the best. However, no hyperparameter tuning has been done. The multiple paths strategy could probably beat the sequential strategy on this dataset with proper regularization and hyperparameters.