Deep Learning
Dr. DavoodAbadi
Fall 2022
Hoorieh Sabzevari - 98412004
HW6

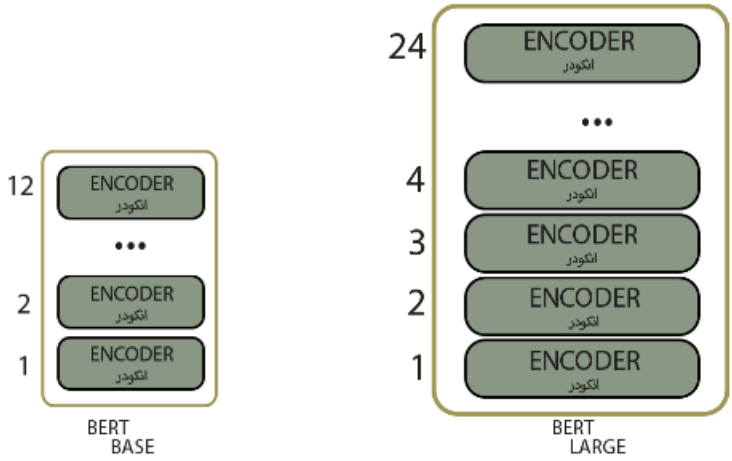
۱. کد مربوطه در فایل Q1.ipynb موجود است.

```
Batch Normalization:
 [[-0.25839035 -1.9522604  -1.32416941 -0.17220346  1.24551176]
 [-0.86130117  0.36725691  1.32416941  1.40632822 -0.92919132]
 [ 1.89486258  0.27061035  0.41380294 -0.74621497  1.19608669]
 [ 0.          0.89881296 -0.99312706 -1.32022649 -0.68206597]
 [-0.77517105  0.41558018  0.57932412  0.8323167  -0.83034118]]

Layer Normalization:
 [[-0.87558997 -1.07958857 -1.87064616  0.11541284  0.77748801]
 [-1.27882219  0.50998969  2.9568278   0.69247702 -1.02301053]
 [ 0.56452511  0.44375726  1.29738363 -0.09442868  0.73656758]
 [-0.70277616  0.87426804 -1.26721191 -0.30427021 -0.81840843]
 [-1.22121759  0.54310591  1.59910075  0.4826355  -0.94116969]]
```

۲. در شبکات کانولوشنی مشاهده کردیم که با استفاده از وزن‌های یک مدل از پیش‌آموخته‌شده دیگر نیازی نبود که تمامی مسیر آموزش را از اول طی کنیم. در این حالت کافی است که یک شبکه را با استفاده از دو روش استخراج ویژگی (Feature Extraction) و تنظیم دقیق (Fine-tuning) برای کار خودمان اختصاصی کنیم. همچنین برای استفاده از مدل‌های از قبل آموزش‌دیده در مسائلی که با متن سروکار دارند، ابتدا به سراغ مدل‌های تعبیهٔ کلمات (Word Embedding) رفتیم که به ما کمک و تغییر محسوسی در دقت شبکه‌ها ایجاد کردند، ولی عمیق نبودند و حاوی

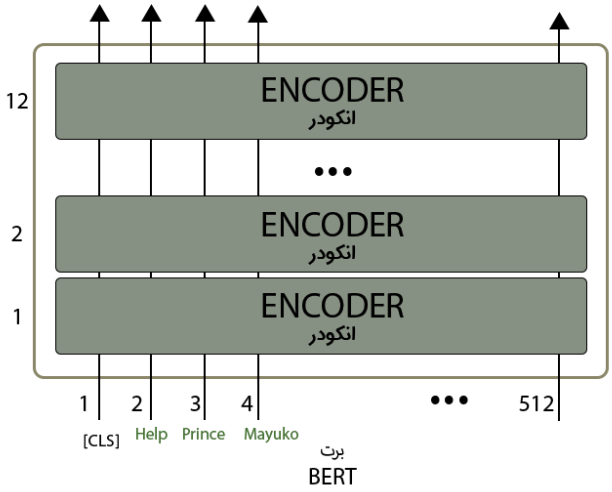
اطلاعات کمی بودند؛ درواقع کمک آن‌ها مؤثر ولی محدود بود. در سال ۲۰۱۸ این مسیر برای مسئله‌های متنی یا به‌طور دقیق‌تر پردازش زبان های طبیعی (Natural Language Processing) نیز در دسترس قرار گرفت. مهندسان گوگل مدل بزرگی با داده‌ی زیاد را آموزش دادند و آن را در دسترس همه قرار دادند. حالا یک مدل بسیار قدرتمند برای بهره‌گیری در مسائل متنی در اختیار داریم؛ این مدل **BERT** نام دارد. مدل BERT یا Bidirectional Encoder Representations from Transform در دو اندازه‌ی متفاوت آموزش داده شده است: BERTBASE و BERTLARGE.

مدل BERT درواقع دسته‌ای از انکودرهای مدل ترنسفورمر (Transformer Model) است که آموزش دیده‌اند. هر دو مدل BERT تعداد زیادی لایه‌ی انکودر دارند. مدل BERTBASE شامل ۱۲ لایه انکودر (در مقاله‌ی اصلی Transformer Blocks نامیده می‌شوند) و مدل بزرگ‌تر که همان مدل BERTLARGE است شامل ۲۴ لایه انکودر است. مدل پایه در مجموع ۱۱۰میلیون پارامتر و مدل بزرگ ۳۴۵میلیون پارامتر دارد. آموزش هر یک از آن‌ها چهار روز زمان برده است. مدل پایه ۷۶۸ و مدل بزرگتر ۱۰۲۴ نود پنهان در لایه‌ی شبکه پیشخور خود دارند و تعداد لایه‌های توجه در اولی ۱۲ و در دومی ۱۶ است.

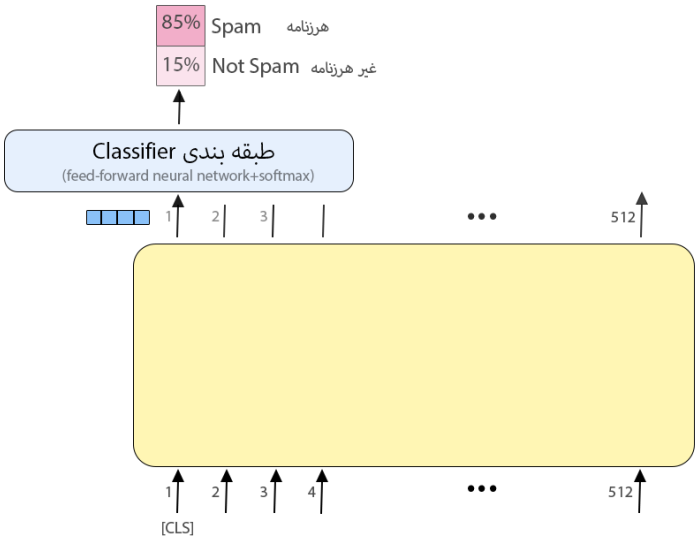


اولین توکن ورودی با یک توکن خاص به‌نام CLS به مدل وارد می‌شود و دقیقاً مانند بخش انکودر مدل ترنسفورمر که در بخش قبلی درباره‌ی آن صحبت شد، مدل BERT توالی از کلمات را در ورودی دریافت می‌کند. این‌ها در طول لایه‌های انکودر موجود حرکت می‌کنند. هر لایه‌ی انکودر یک

لایه‌ی Self-Attention و یک لایه‌ی شبکه‌ی پیشخور را شامل است که ورودی‌ها از آن‌ها می‌گذرند و سپس به لایه‌ی انکودر بعدی وارد می‌شوند. هر موقعیت یک بردار به‌اندازه‌ی نودهای لایه‌ی پنهان را در خروجی ارائه می‌کند؛ برای مثال در مدل BERTBASE اندازه‌ی لایه‌ی پنهان ۷۶۸ است؛ پس در خروجی بردارهایی به‌اندازه‌ی ۷۶۸ خواهیم داشت. در مسئله‌ی طبقه‌بندی فقط بردار خروجی اول محل تمرکز ماست که ورودی آن همان توکن CLS بود.



این بردار خروجی در مسئله‌ی طبقه‌بندی به‌عنوان ورودی به لایه‌ی طبقه‌بندی وارد می‌شود تا نتیجه را در خروجی نمایش دهد.



برای آموزش BERT از این دو روش استفاده شده است:

۱. مدل زبانی نقاب‌دار (Masked Language Model)

۲. پیش‌بینی جمله‌ی بعدی (Next Sentence Prediction / NSP)

به کمک از دو روش تنظیم دقیق و استخراج ویژگی نیز می‌توان از این مدل استفاده کرد. (منبع)

۳. الف) ابتدا هایپرپارامترهای تعداد کلاس‌ها، سایز هر دسته، تعداد ایپاک‌ها و ابعاد ورودی دیتا را تعریف می‌کنیم. سپس هر x را نرمالیزه کرده و yها هم به فرمت categorical در می‌آوریم.

```
CLASSES = 10
BATCH_SIZE = 32
EPOCHS = 30
INPUT_SHAPE = x_train.shape[1:]
```

```
x_train = np.divide(x_train.astype('float32'), 255)
x_test = np.divide(x_test.astype('float32'), 255)
y_train = to_categorical(y_train, num_classes=CLASSES)
y_test = to_categorical(y_test, num_classes=CLASSES)

print("X train:\n", x_train.shape, "\n\nY train:\n", y_train.shape)

print("\nX test:\n", x_test.shape, "\n\nY test:\n", y_test.shape)
```

```
X train:
 (200, 32, 32, 3)

Y train:
 (200, 10)

X test:
 (10000, 32, 32, 3)

Y test:
 (10000, 10)
```

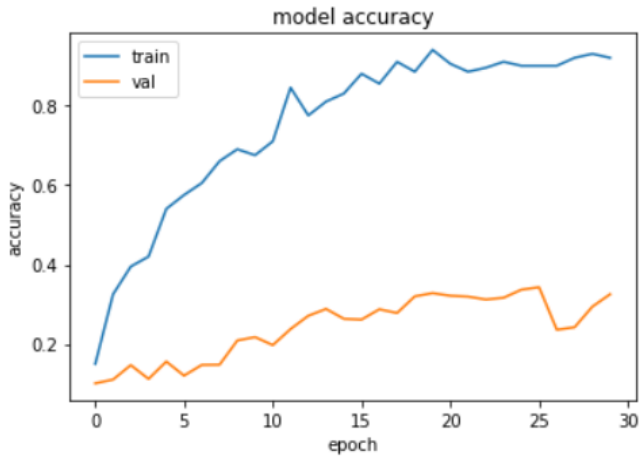
سپس مدل خود را به صورت sequential تعریف می‌کنیم. در لایه‌ی آخر از یک لایه dense با ۱۰نورون برای دسته‌بندی استفاده می‌کنیم.

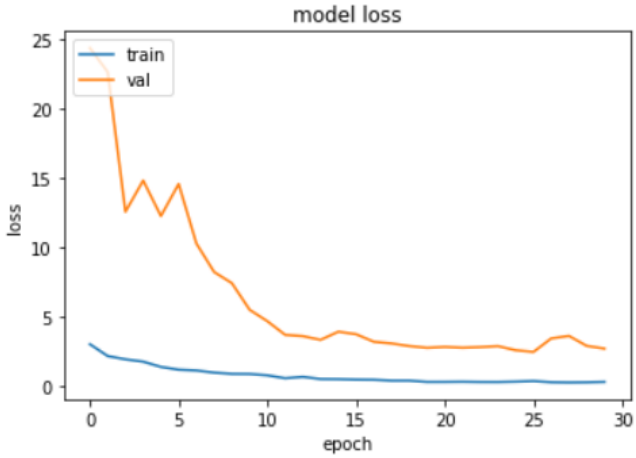
```
# Model
model_a = tensorflow.keras.Sequential([Input(shape=INPUT_SHAPE),
                                       Conv2D(32, (5, 5), padding="same"),
                                       Activation("relu"),
                                       BatchNormalization(axis=-1),
                                       MaxPooling2D(pool_size=(2, 2)),
                                       Dropout(0.25),
                                       Conv2D(64, (5, 5), padding="same"),
                                       Activation("relu"),
                                       BatchNormalization(axis=-1),
                                       MaxPooling2D(pool_size=(2, 2)),
                                       Dropout(0.3),
                                       Conv2D(128, (5, 5), padding="same"),
                                       Activation("relu"),
                                       BatchNormalization(axis=-1),
                                       MaxPooling2D(pool_size=(2, 2)),
                                       Dropout(0.3),
                                       Flatten(),
                                       Dense(512),
                                       Activation("relu"),
                                       BatchNormalization(),
                                       Dropout(0.3),
                                       Dense(10),
                                       Activation("softmax"),
                                       ])
model_a.summary()
```

سپس مدل خود را compile می‌کنیم و با ۳۰ ایپاک و اندازه‌ی دسته ۳۲ آموزش می‌دهیم. از بهینه‌ساز adam و تابع ضرر cce استفاده می‌کنیم.

```
Epoch 27/30
7/7 [==============================] - 1s 167ms/step - loss: 0.2658 - accuracy: 0.9000 - val_loss: 3.4297 - val_accuracy: 0.2359
Epoch 28/30
7/7 [==============================] - 1s 219ms/step - loss: 0.2518 - accuracy: 0.9200 - val_loss: 3.6064 - val_accuracy: 0.2423
Epoch 29/30
7/7 [==============================] - 1s 183ms/step - loss: 0.2620 - accuracy: 0.9300 - val_loss: 2.8864 - val_accuracy: 0.2940
Epoch 30/30
7/7 [==============================] - 1s 219ms/step - loss: 0.2912 - accuracy: 0.9200 - val_loss: 2.6941 - val_accuracy: 0.3253
```

دقت در داده‌های آموزشی به ۹۲٪ و در داده‌های validation به ۳۲٪ رسیده است که نشان می‌دهد مدل ما overfit شده است. زیرا داده‌های آموزشی فقط ۲۰۰ تا بوده و به دلیل تعداد کم مدل آن‌ها را حفظ کرده است.





ب) برای مسئله تشخیص زاویه باید تصاویر بدون برچسب را در چهار زاویه‌ی مختلف بچرخانیم و برای هرکدام برچسب مخصوص خود را در فرمت one-hot تعیین کنیم.

```
x_rotated = []
y_rotated = []

data = np.divide(x_unlabeld.astype('float32'), 255)

for img in data:
  for i in range(4):
    x = np.rot90(img, k=i)
    y = np.zeros(4)
    y[i] = 1
    x_rotated.append(x)
    y_rotated.append(y)

x_rotated = np.array(x_rotated)
y_rotated = np.array(y_rotated)
```

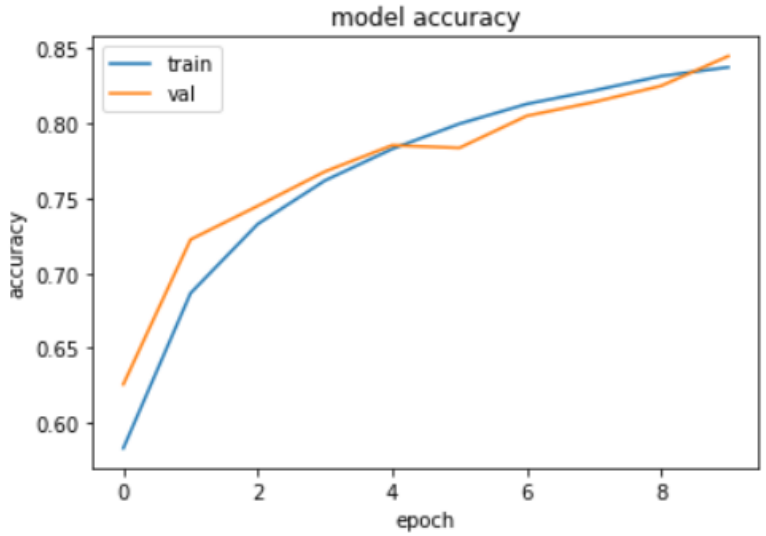
سپس مدل خود را تعریف می‌کنیم.

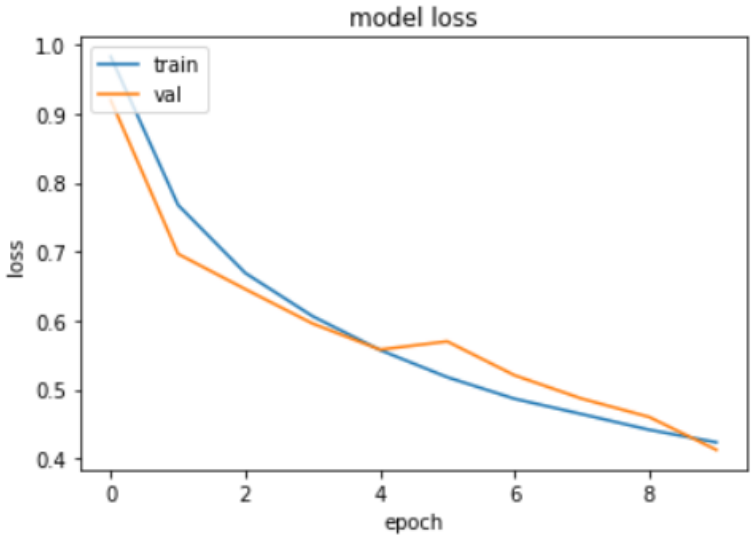
```
# Model
model_ssl = tensorflow.keras.Sequential([
  Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)),
  BatchNormalization(),
  Conv2D(32, (3, 3), activation='relu', padding='same'),
  BatchNormalization(),
  MaxPooling2D((2, 2)),
  Dropout(0.3),
  Conv2D(64, (3, 3), activation='relu', padding='same'),
  BatchNormalization(),
  Conv2D(64, (3, 3), activation='relu', padding='same'),
  BatchNormalization(),
  MaxPooling2D((2, 2)),
  Dropout(0.3),
  Conv2D(128, (3, 3), activation='relu', padding='same'),
  BatchNormalization(),
  Conv2D(128, (3, 3), activation='relu', padding='same'),
  BatchNormalization(),
  MaxPooling2D((2, 2)),
  Dropout(0.3),
  Flatten(),
  Dense(64, activation='relu', kernel_initializer='he_uniform'),
  Dense(4, activation='softmax')])

model_ssl.summary()
```

مدل خود را کامپایل می‌کنیم و با همان تابع بهینه‌ساز و ضرر آموزش می‌دهیم، اما به دلیل اینکه ram زود پر می‌شد تعداد ایپاک‌ها را به ۱۰ کاهش داده و سایز هر دسته را نیز تا ۱۲۸ افزایش دادیم. همچنین ۲۰٪ از داده‌ها را نیز به عنوان validation جدا می‌کنیم.

```
Epoch 7/10
1245/1245 [==============================] - 20s 16ms/step - loss: 0.4868 - accuracy: 0.8129 - val_loss: 0.5208 - val_accuracy: 0.8050
Epoch 8/10
1245/1245 [==============================] - 21s 17ms/step - loss: 0.4645 - accuracy: 0.8219 - val_loss: 0.4869 - val_accuracy: 0.8142
Epoch 9/10
1245/1245 [==============================] - 21s 17ms/step - loss: 0.4419 - accuracy: 0.8316 - val_loss: 0.4604 - val_accuracy: 0.8251
Epoch 10/10
1245/1245 [==============================] - 20s 16ms/step - loss: 0.4237 - accuracy: 0.8375 - val_loss: 0.4127 - val_accuracy: 0.8449
```

همانطور که واضح است دقت ما در این تسک ساختگی به ۸۳٪ در داده‌های آموزشی و ۸۴٪ در داده‌های validation رسیده است. حال از وزن‌های بدست‌آمده در مسئله‌ی Downstreem استفاده خواهیم کرد.





حال مدل خود را با استفاده از وزن‌های جدید می‌سازیم.

```
# Model
model_b = keras.Model(model_ssl.inputs, Dense(10 ,activation='softmax')(model_ssl.layers[-2].output))
model_b.summary()
```

در لایه‌ی آخر بجای ۴ نورون از ۱۰ نورون برای دسته‌بندی استفاده می‌کنیم. سپس دوباره مدل را آموزش می‌دهیم.

```
Epoch 28/30
7/7 [==============================] - 1s 184ms/step - loss: 0.0932 - accuracy: 0.9800 - val_loss: 2.9572 - val_accuracy: 0.3663
Epoch 29/30
7/7 [==============================] - 1s 178ms/step - loss: 0.1065 - accuracy: 0.9750 - val_loss: 2.9618 - val_accuracy: 0.3650
Epoch 30/30
7/7 [==============================] - 1s 221ms/step - loss: 0.1404 - accuracy: 0.9650 - val_loss: 2.9638 - val_accuracy: 0.3699
```

همانگونه که واضح است، دوباره مدل دچار overfit شد اما دقت ما در داده‌های validation افزایش یافته است و به دقت ۳۷٪ رسیده است. قطعا اگر کار بیشتری روی مدل انجام دهیم و پارامترهای آن را تغییر دهیم به دقت بسیار بهتری هم در داده‌های آموزشی و هم در داده‌های validation خواهیم رسید. چرا که وزن‌های ما در تسک ساختگی نتایج خوبی را داشته‌اند.

پ) در این قسمت طبق خواسته‌ی سوال یک مدل با دو خروجی می‌سازیم.

```
# Model
base_model = tensorflow.keras.Sequential([model_b])

classifier = Dense(units=10, activation='softmax', name='classifier')(base_model.outputs[0])
rotator = Dense(units=4, activation='softmax', name='rotator')(base_model.outputs[0])

model_c = keras.Model(inputs=base_model.inputs, outputs=[classifier, rotator])

model_c.summary()
```

سپس تمام دیتاها را باهم concatenate می‌کنیم. برای هر ورودی هم دو نوع خروجی، یکی برای دسته‌بندی بین ۱۰ کلاس و دیگری برای دسته‌بندی بین ۴ کلاس چرخش قرار می‌دهیم.

```
x_rotated = np.zeros_like(x_unlabeld)
y_rotated = np.zeros((x_unlabeld.shape[0], 4))

y_train_class = np.concatenate((y_train, np.zeros((x_rotated.shape[0], 10))), axis=0)
y_train_rot = np.concatenate((np.zeros((y_train.shape[0], 4)), y_rotated), axis=0)

x_train = np.concatenate((x_train, x_rotated), axis=0)
y_train = [y_train_class, y_train_rot]

print(x_train.shape)
print(y_train[0].shape)
print(y_train[1].shape)
```

```
(50000, 32, 32, 3)
(50000, 10)
(50000, 4)
```

هنگام آموزش از سه وزن مختلف برای دو کلاس استفاده می‌کنیم و حالات مختلف را بررسی می‌کنیم.

حالت اول: ابتدا وزن دسته‌بند را بیشتر از وزن چرخش قرار می‌دهیم.

```
classifier_accuracy: 0.1003 - rotator_accuracy: 0.2480
```

حالت دوم: وزن چرخش را بیشتر از وزن دسته‌بند قرار می‌دهیم.

```
classifier_accuracy: 0.1015 - rotator_accuracy: 0.2562
```

حالت سوم: وزن هر دو را برابر قرار می‌دهیم.

```
classifier_accuracy: 0.1037 - rotator_accuracy: 0.2523
```

چون تعداد داده‌های آموزشی برچسب‌دار برای تسک دسته‌بندی ۱۰ کلاسه بسیار کم بوده لذا از دقت کمتری برخوردار است و برای اینکه هر تسک به دقت خوبی برسد باید وزن آن را بیشتر کنیم. در مجموع اگر بخواهیم دو تسک باهم خوب پیش بروند باید هر دو وزن را یکسان در نظر بگیریم.

۴. الف) مدل داده‌شده را با استفاده از ابزار keras tuner به صورت زیر بازنویسی کردیم و حداقل سه مقدار برای هایپرپارامترهای خواسته‌شده در صورت سوال انتخاب کردیم.

```
def build_model(hp):
  inputs = Input(shape = (32, 32, 3))
  x = inputs

  for i in range(hp.Int('conv_blocks',min_value = 3, max_value = 5, default=3)):
    filters = hp.Int('filters_' + str(i),min_value = 32,max_value = 128, step=32)
    for _ in range(2):
      x = Convolution2D(filters, kernel_size=(3, 3), padding= 'same')(x)
      x = BatchNormalization()(x)
      x = ReLU()(x)
    if hp.Choice('pooling_' + str(i), ['avg', 'max']) == 'max':
        x = MaxPool2D()(x)
    else:
        x = AvgPool2D()(x)
    hp_dropout = hp.Choice('rate', values=[0.5, 0.3, 0.4])
    x = Dropout(rate=hp_dropout)(x)

  x = GlobalAvgPool2D()(x)
  x = Dense(hp.Int('Dense units',min_value = 64, max_value = 256, step=32, default=50), activation='relu')(x)
  outputs = Dense(10, activation= 'softmax')(x) # output layer

  model = Model(inputs, outputs)

  hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
  model.compile(optimizer=keras.optimizers.Adam(learning_rate=hp_learning_rate),
                loss=keras.losses.CategoricalCrossentropy(),
                metrics=['accuracy'])
  return model
```

برای تعداد بلاک‌های کانولوشنی مقادیر ۳و۴و۵،

برای تعداد فیلترها مقادیر ۳۲ تا ۱۲۸ با گام‌های ۳۲تایی،

برای pooling دو نوع max و avg،

برای نرخ dropout مقادیر بین ۰.۳ و ۰.۴ و ۰.۵،

برای نورون‌های لایه‌ی ماقبل آخر مقادیر ۶۴ تا ۲۵۶ با گام‌های ۳۲تایی

و برای نرخ آموزش مقادیر 0.0001 , 0.001 , 0.01 را انتخاب کردیم.

ب) سپس با استفاده از ابزار keras tuner ابتدا بین گزینه‌های انتخابی و با تعداد ایپاک ۵ سرچ انجام داده و سپس بهترین مدل و بهترین هایپرپارامترها را بدست می‌آوریم.

```
tuner = keras_tuner.RandomSearch(build_model, objective='val_accuracy', max_trials=5)
```

```
tuner.search(img_train, label_train, epochs=5, validation_data=(img_test, label_test), callbacks=[tf.keras.callbacks.EarlyStopping(patience=2)])
```

```
Trial 5 Complete [00h 03m 03s]
val_accuracy: 0.7055000066757202

Best val_accuracy So Far: 0.7458999752998352
Total elapsed time: 00h 12m 59s
```

```
# Get the optimal hyperparameters
best_hps= tuner.get_best_hyperparameters(1)[0]

# get the best model
best_model = tuner.get_best_models(1)[0]
```

بهترین هایپرپارامترها به صورت زیر و بهترین مدل نیز در فایل کد موجود است.

```
Number of conv blocks: 4
filters_0: 96
filters_1: 64
filters_2: 64
filters_3: 96
pooling_0: avg
pooling_1: max
pooling_2: avg
pooling_3: avg
rate: 0.3
Dense units: 64
learning_rate: 0.001
```

همانطور که می‌بینیم این ابزار اعداد خوبی را گزارش کرده است. طبق این مدل بهتر است ۴ بلاک کانولوشنی داشته باشیم و تعداد بیشتری از بلاک‌ها باعث overfit شدن مدل می‌شود. تعداد فیلترها نیز به صورت صعودی بالا رفته که منطقی است. از نرخ آموزش ۰.۰۰۱ استفاده کرده که هم باعث همگرایی سریع‌تر مدل می‌شود. مقدار ۰.۳ برای احتمال dropout در نظر گرفته شده و از pooling ها در جای مناسبی استفاده شده است و باعث دقت بیشتری در داده‌های validation شده است. در لایه‌ی ماقبل آخر نیز بهتر است ۶۴ نورون داشته باشیم.

پ) سپس مدل را با همان تعداد ایپاک و batch_size آموزش می‌دهیم.

```
Epoch 28/30
782/782 [==============================] - 14s 18ms/step - loss: 0.2649 - accuracy: 0.9063 - val_loss: 0.4812 - val_accuracy: 0.8502
Epoch 29/30
782/782 [==============================] - 14s 18ms/step - loss: 0.2660 - accuracy: 0.9063 - val_loss: 0.5168 - val_accuracy: 0.8391
Epoch 30/30
782/782 [==============================] - 14s 18ms/step - loss: 0.2555 - accuracy: 0.9100 - val_loss: 0.5293 - val_accuracy: 0.8421
```

که این بار با مدل بهتر به دقت ۹۱٪ در داده‌های آموزشی و دقت ۸۴٪ در داده‌های validation رسیدیم.

پارامترهای خواسته‌شده در صورت سوال به شرح زیر است:

```
y_pred = best_model.predict(img_test)
y_pred=np.argmax(y_pred, axis=1)
# label_test=np.argmax(label_test, axis=1)

print(classification_report(label_test, y_pred))
```

```
313/313 [==============================] - 1s 3ms/step
              precision    recall  f1-score   support

           0       0.87      0.85      0.86      1000
           1       0.91      0.92      0.91      1000
           2       0.89      0.70      0.78      1000
           3       0.65      0.78      0.71      1000
           4       0.80      0.87      0.83      1000
           5       0.90      0.66      0.76      1000
           6       0.78      0.94      0.85      1000
           7       0.95      0.83      0.89      1000
           8       0.90      0.93      0.91      1000
           9       0.87      0.94      0.90      1000

    accuracy                           0.84     10000
   macro avg       0.85      0.84      0.84     10000
weighted avg       0.85      0.84      0.84     10000
```

همانطور که می‌بینیم، هر ۴معیار تقریبا ۸۴٪ هستند و مدل نسبتا خوب عمل کرده است. معیار F1 که در واقع ترکیب متعادلی بین معیارهای دقت و صحت است، می‌تواند در مواردی که هزینه‌ی False Positive و False Negative متفاوت است به کار رود. اگر هزینه‌ی False Positive و False Negative تقریباً برابر بود، می‌توان از همان معیار دقت (Accuracy) استفاده کرد. همچنین اگر داده‌های ما در کلاس‌ها به صورت نامتوازن پخش شده بود (برای مثال اگر ۹۰ درصد بیمار و ۱۰ درصد شخص سالم داشتیم) بهتر است که از معیارهای صحت، پوشش و یا F1 استفاده کنیم. در این سوال چون دیتا بالانس است و تعداد نمونه‌ها از هر کلاس تقریبا متوازن است، لذا استفاده از دقت معیار خوبی است.

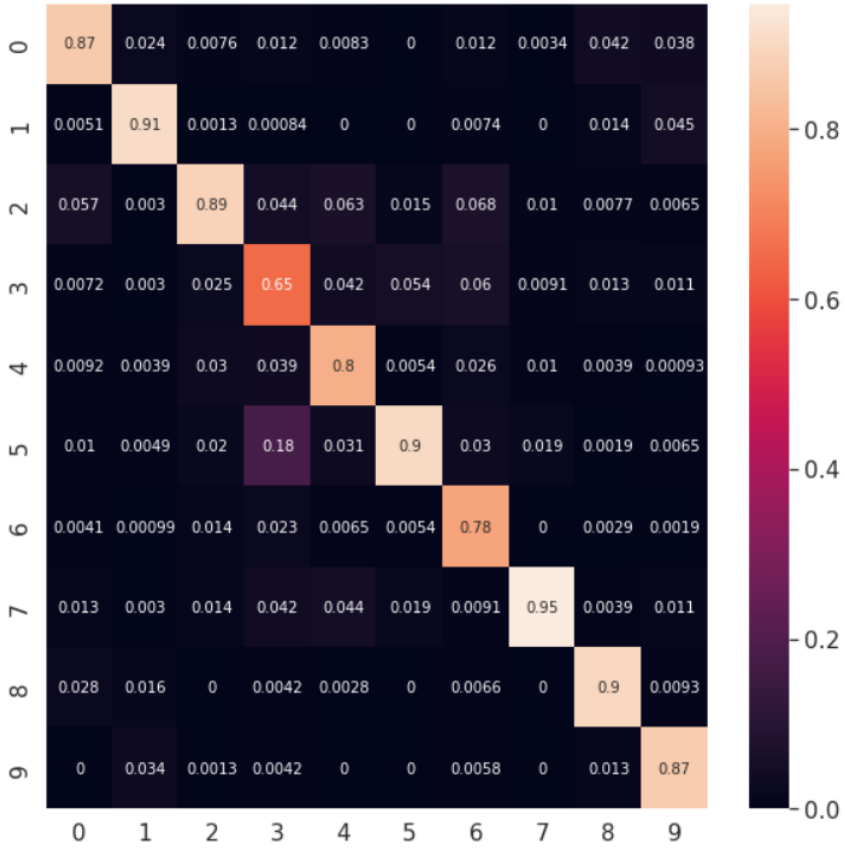
ت) مثبت صحیح : (TP , True Positive) وقتی نمونه عضو دسته مثبت باشد و به عنوان عضو همین دسته تشخیص داده شود.

منفی کاذب : (FN , False Negative) وقتی نمونه عضو دسته مثبت باشد و به عنوان عضو دسته منفی تشخیص داده شود.

منفی صحیح : (TN , True Negative) وقتی نمونه عضو دسته منفی باشد و به عنوان عضو همین دسته تشخیص داده شود.

مثبت کاذب (FP , False Positive) : وقتی نمونه عضو دسته منفی باشد و به عنوان عضو دسته مثبت تشخیص داده شود .

نقشه‌ی confusion matrix عملکرد مدل را بر اساس چهار مفهوم بالا نمایش می‌دهد و طبق نقشه‌ی زیر، مدل در کلاس‌های ۳،۴،۶ نسبتا ضعیف عمل کرده است.



لینک‌های استفاده‌شده در این سوال:

https://blog.paperspace.com/hyperparameter-optimization-with-keras-/tuner

https://datascience.stackexchange.com/questions/45165/how-to-get-accuracy-f1-precision-and-recall-for-a-keras-model

/https://chistio.ir/precision-recall-f

https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

https://stackoverflow.com/questions/70775762/how-to-make-a-confusion-matrix-with-keras