به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اکسترا

نام و نام خانوادگی	سارا رستمی – امین شاهچراغی
شماره دانشجویی	۸۱۰۱۰۰۳۵۵ - ۸۱۰۱۹۹۱۹۶
تاریخ ارسال گزارش	۱۴۰۱.۰۹.۲۴

فهرست

پاسخ ۱. **تشخیص تقلب (fraud detection) با استفاده از شبکه عمیق** ۱

۱. ..... ۱

۲. ..... ۱

۳. ..... ۱

۴. ..... ۲

۵. ..... ۴

۶. ..... ۵

۷. ..... ۶

پاسخ ۲. تشخیص زنده بودن ۷

الف. ۷

ب. ۷

پ. ۹

ت. ۱۰

ث. ۱۱

پاسخ ۳ . **تشخیص کاراکتر نوری (Optical character recognition)** ۱۲

الف. ۱۲

ب. ۱۲

ج. ۱۳

د. ۱۵

ه. ۲۱

شکل‌ها

شکل ۱- خواندن داده‌ها و پیش‌پردازش آنها ۲
شکل ۲- انجام روش SMOTE و مقایسه ابعاد داده قبل و بعد ۳
شکل ۳ - اضافه کردن نویز به داده تست ۳
شکل ۴- طراحی و آموزش auto-encoder ۳
شکل ۵- شبکه‌ی fully-connected ۴
شکل ۶ - ماتریس آشفتگی مدل ۴
شکل ۷- نتایج مدل ۵
شکل ۸ - نمودار precision و recall برای مدل اصلی ۵
شکل ۹- ماتریس آشفتگی مدل بدون oversampling و noise ۶
شکل ۱۰- نتایج مدل بدون oversampling و recall ۶
شکل ۱۱- نمودار precision و recall برای مدل بدون oversampling و noise ۶
شکل ۱۲- خواندن داده‌ها و تقسیم آن‌ها به train و validation ۷
شکل ۱۳- معماری مدل ۸
شکل ۱۴- نمودار دقت مدل روی داده‌های train و validation ۹
شکل ۱۵- نمودار loss مدل روی داده‌های train و validation ۹
شکل ۱۶- معماری LeNet-5 ..... ۱۰
شکل ۱۷- معماری AlexNet ۱۰
شکل ۱۸- نمودار loss مدل طی ۲۰ ایپاک ۱۱
شکل ۱۹- نمودار دقت مدل طی ۲۰ ایپاک ۱۱
شکل ۲۰- خواندن داده‌ها و انجام پیش‌پردازش‌ها ۱۳
شکل ۲۱- معماری مدل مقاله ۱۵
شکل ۲۲- آموزش مدل با بهینه‌ساز momentum ۱۶
شکل ۲۳- نمودار دقت مدل با بهینه‌ساز momentum ۱۶
شکل ۲۴- نمودار loss مدل با بهینه‌ساز momentum ۱۶
شکل ۲۵- نتیجه‌ی مدل با بهینه‌ساز momentum ۱۷
شکل ۲۶- ماتریس آشفتگی مدل با بهینه‌ساز momentum ۱۷
شکل ۲۷- آموزش مدل با بهینه‌ساز Adam ۱۷

شکل ۲۸- نمودار دقت مدل با بهینه‌ساز Adam ۱۸
شکل ۲۹- نمودار loss مدل با بهینه‌ساز Adam ۱۸
شکل ۳۰- نتیجه‌ی مدل با بهینه‌ساز Adam ۱۸
شکل ۳۱- ماتریس آشفتگی مدل با بهینه‌ساز Adam ۱۹
شکل ۳۲- آموزش مدل با بهینه‌ساز AdaDelta ۱۹
شکل ۳۳- نمودار دقت مدل با بهینه‌ساز AdaDelta ۱۹
شکل ۳۴- نمودار loss مدل با بهینه‌ساز AdaDelta ۲۰
شکل ۳۵- نتیجه‌ی مدل با بهینه‌ساز AdaDelta ۲۰
شکل ۳۶- ماتریس آشفتگی مدل با بهینه‌ساز AdaDelta ۲۰

جدول‌ها

جدول ۱- مقایسه‌ی ۳ بهینه‌ساز ۲۱

پاسخ ۱. تشخیص تقلب (fraud detection) با استفاده از شبکه عمیق

.۱

بزرگترین مشکل پیش روی مدل های تشخیص تقلب نبود داده های مناسب و بالانس برای آموزش مدل های یادگیری عمیق می باشد. معمولا در مسائل کاربردی این حوزه با داده هایی روبرو هستیم که کلاس تقلب آن تعداد بسیار کمی دارد و از طرحی اهمیت بالایی دارد که این داده ها حتما تشخیص داده شوند. کلاسبندی اشتباه اغلب در کلاس اقلیت اتفاق می افتد، زیرا مدل طبقه بندی سعی خواهد کرد تمام نمونه داده ها را به کلاس اکثریت طبقه بندی کند. در این مقاله، نمونه برداری بیش از حد یا Oversampling تکنیکی است که برای مقابله با آن استفاده شده است. روش پیشنهادی مقاله SMOTE یا Synthetic Minority Oversampling Technique می باشد. در این روش به منظور ایجاد یک نقطه داده مصنوعی، ابتدا ما باید k نزدیکترین خوشه همسایه را در فضا پیدا کنیم سپس یک نقطه تصادفی را در خوشه انتخاب می کنیم و از میانگین وزنی برای جعل نقطه جدید استفاده می کنیم.

.۲

در این مقاله برای حل مشکل بالانس نبودن داده ها و robust کردن شبکه معماری زیر پیشنهاد شد. بعد از اعمال SMOTE داده ها نویزی می شوند و سپس به اولین قسمت شبکه که اتوانکودر denoised می باشد وارد می شوند که شامل ۶ لایه می باشد و تابع محاسبه خطا آن مربع خطا می باشد.

سپس داده نویز زدایی شده برای کلاس بندی وارد یک شبکه 5 fully connected لایه می شود که از softmax و cross-entropy استفاده می کند.

.۳

دو روش برای مقابله با داده های نامتوازن وجود دارد:

۱.در نمونه گیری Oversampling سعی می شود از کلاس اقلیت نمونه های بیشتر ایجاد شود تا نسبت کلاس ها به هم نزدیک شود. در این روش اطلاعات ارزشمند از بین نمی روند و همه داده ها حفظ می شوند گرچه با افزایش هزینه محاسباتی رو به رو می شویم. افزایش تعداد نمونه ها در کلاس اقلیت (به ویژه برای یک مجموعه داده به شدت نا متوازن) ممکن است منجر به افزایش بار محاسباتی بشود.

۲. در روش دوم که Undersampling است سعی می شود از کلاس حداکثر نمونه گیری کنیم. در واقع در این روش ما از همه نمونه ها در کلاس بیشتر استفاده نمی کنیم تا نسبت کلاس ها به یکدیگر نزدیک شود. در اینجا گرچه محاسبات زیادی نداریم اما در واقع تعدادی از داده هارا حذف می کنیم که می تواند برای ما ارزشمند باشد.

ترکیب هر دو روش نمونه گیری Undersampling و Oversampling تصادفی می تواند منجر به بهبود عملکرد کلی در مقایسه با روش های جداگانه شود.

به این معنی که ما می توانیم مقدار کمی از کلاس اکثریت حذف کنیم و تعداد کم تری به کلاس اقلیت اضافه کنیم. این روش سعی می کند معایب روش های قبلی را نداشته باشد و از محاسن آن ها استفاده کند.

۴.

مدل ارائه شده پیاده سازی شد. ابتدا داده ها را خواندیم و پیش پردازش های لازم مثل نرمال سازی داده های Time را انجام دادیم. داده هارا به داده آموزش و ارزیابی تقسیم کردم.

```
df = pd.read_csv('drive/My Drive/creditcard.csv')
del df["Time"]
df["Amount"] = (df['Amount'] - df["Amount"].min()) / (df["Amount"].max() - df["Amount"].min())
```

```
X=df.copy()

y=df.copy()


del X["Class"]
y=y["Class"]
y = np.expand_dims(y, axis=1)
# split into 70:30 ration
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)

# describes info about train and test set
print("Number transactions X_train dataset: ", X_train.shape)
print("Number transactions y_train dataset: ", y_train.shape)
print("Number transactions X_test dataset: ", X_test.shape)
print("Number transactions y_test dataset: ", y_test.shape)
```

```
Number transactions X_train dataset:  (199364, 29)
Number transactions y_train dataset:  (199364, 1)
Number transactions X_test dataset:  (85443, 29)
Number transactions y_test dataset:  (85443, 1)
```

شکل ۱- خواندن داده‌ها و پیش‌پردازش آنها

در ادامه به روش SMOTE تکنیک Oversampling را اجرا کردم.

```
sm = SMOTE(random_state = 2)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train.ravel())
y_train_res = np.expand_dims(y_train_res, axis=1)

print('After OverSampling, the shape of train_X: {}'.format(X_train_res.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_res.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train_res == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_res == 0)))
```

```
Before OverSampling, counts of label '1': [345]
Before OverSampling, counts of label '0': [199019]

After OverSampling, the shape of train_X: (398038, 29)
After OverSampling, the shape of train_y: (398038, 1)

After OverSampling, counts of label '1': [199019]
After OverSampling, counts of label '0': [199019]
```

شکل ۲- انجام روش **SMOTE** و مقایسه ابعاد داده قبل و بعد

در ادامه نویز Gaussian را به داده های آموزش و تست اضافه کردم.

Adding Noise

```
mu, sigma = 0, 0.1
noise = np.random.normal(mu, sigma, [85443,29])
noise
```

```
array([[ 0.05374348, -0.17733136,  0.06822311, ...,  0.12001667,
         0.1532451 ,  0.03305445],
       [-0.049773  , -0.20206519, -0.00271666, ...,  0.08823793,
         0.03190902,  0.07699753],
       [ 0.01596794,  0.03699334,  0.00737976, ..., -0.02564186,
        -0.08467136,  0.00138765],
       ...,
       [-0.0583132 ,  0.04016636, -0.02823627, ..., -0.0682999 ,
         0.01101897,  0.14551142],
       [ 0.11143277, -0.04420132, -0.20710557, ..., -0.0152895 ,
        -0.0723689 , -0.04770944],
       [-0.03119636,  0.02192217,  0.1784115 , ...,  0.16216124,
        -0.15064081, -0.04481508]])
```

```
X_test_noised=X_test+noise
```

شکل ۳ - اضافه کردن نویز به داده تست

در مرحله بعد اتوانکودر گفته شده در مقاله را پیاده سازی کردم و آن را برای ۶۰ epoch آموزش دادم.

```
# Building the Input Layer
input_layer = Input(shape =(X.shape[1], ))

# Building the Encoder network
encoded = Dense(22, activation ='leaky_relu')(input_layer)
encoded = Dense(15, activation ='leaky_relu')(input_layer)
# Building the Decoder network
decoded = Dense(10, activation ='leaky_relu')(encoded)
decoded = Dense(15, activation ='leaky_relu')(decoded)
decoded = Dense(22, activation ='leaky_relu')(decoded)

# Building the Output Layer
output_layer = Dense(X.shape[1], activation ='leaky_relu')(decoded)
```

```
# Defining the parameters of the Auto-encoder network
autoencoder = Model(input_layer, output_layer)
autoencoder.compile(optimizer ="Adam", loss ="mse")

# Training the Auto-encoder network
autoencoder.fit(X_train_noised, X_train_res, batch_size = 32, epochs = 60, verbose=1, validation_data=(X_test_noised, X_test))
```

```
12439/12439 [==============================] - 30s 2ms/step - loss: 0.2225 - val_loss: 0.2645
Epoch 33/60
12439/12439 [==============================] - 31s 2ms/step - loss: 0.2223 - val_loss: 0.2647
Epoch 34/60
12439/12439 [==============================] - 31s 2ms/step - loss: 0.2218 - val_loss: 0.2626
Epoch 35/60
12439/12439 [==============================] - 31s 2ms/step - loss: 0.2219 - val_loss: 0.2615
```

شکل ۴- طراحی و آموزش **auto-encoder**

در مرحله بعد شبکه عصبی معرفی شده توسط مقاله را پیاده سازی کردم و آن را برای ۳۵ مرحله آموزش دادم. برای جلوگیری از بیش برازش همواره بهترین نتیجه مدل برای داده ارزیابی به عنوان معیار قرار داده شد.

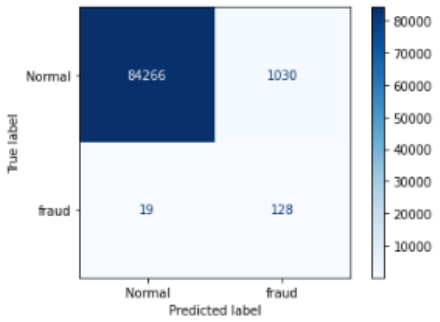
```
# Building the Classifier network
model = Sequential()
model.add(Dense(22, input_dim=29, activation='leaky_relu'))
model.add(Dense(15, activation='leaky_relu'))
model.add(Dense(10, activation='leaky_relu'))
model.add(Dense(5, activation='leaky_relu'))
model.add(Dense(2, activation='leaky_relu'))



model.compile(optimizer ="Adam", loss ="binary_crossentropy")
fname = "weights.{epoch:02d}-{val_loss:.2f}.hdf5"
checkpoint = tf.keras.callbacks.ModelCheckpoint(fname, monitor="val_loss",mode="min", save_best_only=True, verbose=1)
# Training the network
model.fit(X_train_denoised, y_train_res,
                batch_size = 16, epochs = 35,callbacks=[checkpoint], validation_data=(X_test_denoised, y_test))
```

```
24878/24878 [==============================] - 64s 3ms/step - loss: 0.0392 - val_loss: 0.0473
Epoch 22/35
24868/24878 [============================>.] - ETA: 0s - loss: 0.0440
Epoch 22: val_loss did not improve from 0.04611
24878/24878 [==============================] - 64s 3ms/step - loss: 0.0440 - val_loss: 0.0526
Epoch 23/35
24870/24878 [============================>.] - ETA: 0s - loss: 0.0401
Epoch 23: val_loss did not improve from 0.04611
24878/24878 [==============================] - 67s 3ms/step - loss: 0.0400 - val_loss: 0.0527
Epoch 24/35
24876/24878 [============================>.] - ETA: 0s - loss: 0.0419
Epoch 24: val_loss did not improve from 0.04611
24878/24878 [==============================] - 68s 3ms/step - loss: 0.0419 - val_loss: 0.0560
```

شکل ۵- شبکه‌ی **fully-connected**

۵.

در مسائلی که تعداد ها بالانس نیست استفاده از معیار accuracy به تنهایی کافی نیست چون اگر مدل تمام داده هارا به کلاس اکثریت نسبت دهد accuracy مدل بسیار بالا می شود در حالی که می دانیم نتیجه کاملا نامطلوب است. یکی از بهترین معیار ها در این مسائل recall می باشد که اطلاعات لازم را در مورد تشخیص کلاس اقلیت به ما می دهد. هرچه مواردی که ما انتظار داشتیم پیش بینی شوند ولی مدل پیش بینی نکرده‌است که به آن False Negative می‌گوییم نسبت به پیش بینی‌های درست یا True Positive بیشتر باشد مقدار Recall کمتر خواهد شد.



شکل ۶ - ماتریس آشفتگی مدل

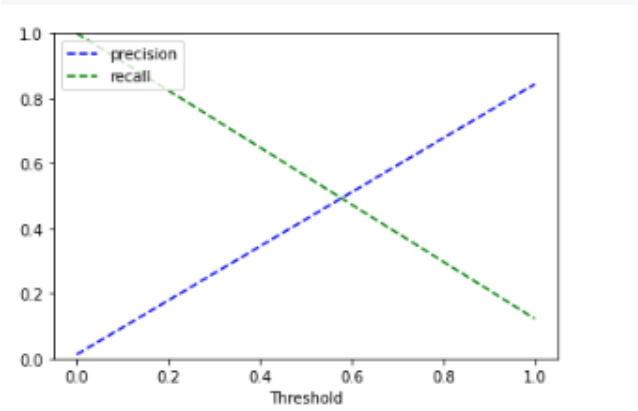
```
print(classification_report(tf.argmax(y_test, axis=1),tf.argmax(pred, axis=1)))
```

	precision	recall	f1-score	support
0	1.00	0.99	0.99	85296
1	0.11	0.87	0.20	147
accuracy			0.99	85443
macro avg	0.56	0.93	0.59	85443
weighted avg	1.00	0.99	0.99	85443

شکل ۷- نتایج مدل

همانطور که مشخص است مدل در تشخیص داده های تقلب خوب عمل می کند و از حدود ۱۴۰ داده تقلب که در داده تست موجود بوده است ۱۲۸ نمونه را درست تشخیص داده است. با ادامه آموزش مدل برای مراحل بیشتر نتیجه قطعا بهتر خواهد شد و مسئله مهم برای ما که از دست ندادن داده تقلب می باشد به وقوع می پیوندد.

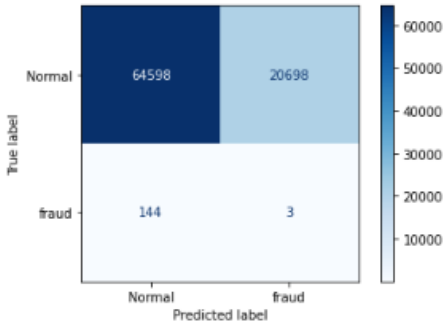
۶.



شکل ۸ - نمودار **precision** و **recall** برای مدل اصلی

۷.

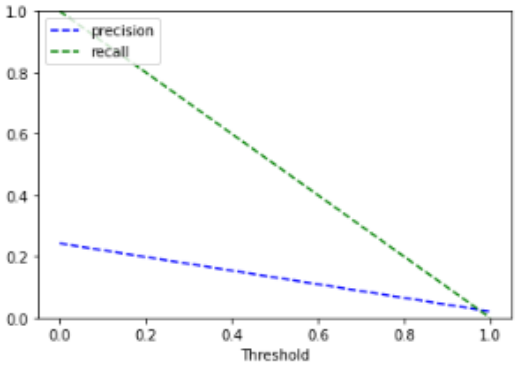
در نهایت یکبار دیگر مدل را با داده های اصلی بدون انجام SMOTE و اضافه کردن نویز آموزش دادم:



شکل ۹- ماتریس آشفتگی مدل بدون **oversampling** و **noise**

	precision	recall	f1-score	support
0	1.00	0.76	0.86	85296
1	0.00	0.02	0.00	147
accuracy			0.76	85443
macro avg	0.50	0.39	0.43	85443
weighted avg	1.00	0.76	0.86	85443

شکل ۱۰- نتایج مدل بدون **oversampling** و **recall**



شکل ۱۱- نمودار **precision** و **recall** برای مدل بدون **oversampling** و **noise**

همانطور که مشخص است مدلی که با داده های Oversample و نویزی شده آموزش دیده است بسیار برای مسائلی با این شرایط که از دست دادن یک داده مثبت برای ما هزینه بالایی دارد، قابل اطمینان تر و بهتر می باشد.

پاسخ ۲. تشخیص زنده بودن

الف.

کاربرد اصلی تشخیص زنده بودن برای مقابله با حمله های حضور یا presentation می باشد.وظیفه کلی تشخیص زنده بودن این است که تشخیص دهد آیا یک کاوشگر بیومتریک (مثلاًاثر انگشت) متعلق به یک سوژه زنده است که در ثبت بیومتریک حضور دارد. با استفاده از تکنیک های تشخیص زنده بودن، تشخیص مطمئن انگشتان غیر زنده یا چهره های عکس گرفته شده را تشخیص داد.

انواع راه حل‌ها با پایه بیومتریک:

اسکنر اثر انگشت – اسکنر رگ – اسکنر صورت – اسکنر صورت و رگ و انگشت (قوی ترین)

با استفاده از ویژگی‌های زیر در اثر انگشت می‌توان به زنده بودن پی برد:

محاسبه پالس و دمای انگشت – تشخیص قطرات عرق و تشخیص مقدار مقاومت پوست

ب.

در مرحله اول داده‌ها در دو دسته آموزش و ارزیابی خواندم.

```
base_path = 'drive/My Drive/Extra/Q2/data/'
image_gen = ImageDataGenerator(rescale=1./255.)
batch_size = 16
train_flow = image_gen.flow_from_directory(
    base_path + 'train/',
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='binary'
)
valid_flow = image_gen.flow_from_directory(
    base_path + 'valid/',
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='binary'
)
```

```
Found 480 images belonging to 2 classes.
Found 120 images belonging to 2 classes.
```

شکل ۱۲- خواندن داده‌ها و تقسیم آن‌ها به **train** و **validation**

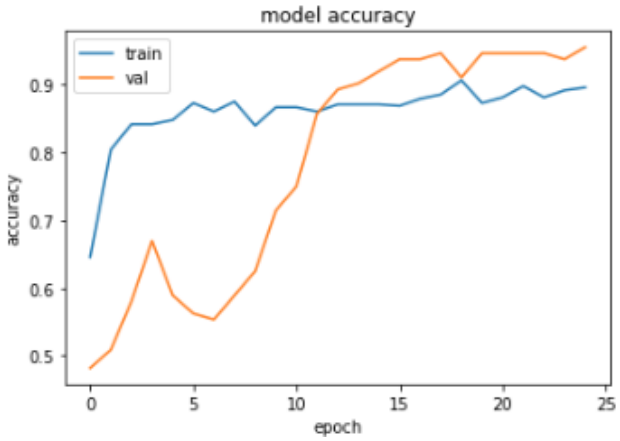
در ادامه با تست و ارزیابی مدل به بهترین معماری و تعداد لایه و پارامتر ها رسیدم که به شرح زیر می باشد. بهینه ساز من Adam و فعال ساز سیگموید بود. از سه لایه کانولوشنی استفاده کردم که بین هر لایه از pooling استفاده شد و همچنین از دو لایه dropout برای جلوگیری از بیش برازش استفاده کردم.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization (BatchN  (None, 224, 224, 3)       12
ormalization)

conv2d (Conv2D)              (None, 224, 224, 8)       224

max_pooling2d (MaxPooling2D  (None, 112, 112, 8)       0
)

batch_normalization_1 (Batc  (None, 112, 112, 8)       32
hNormalization)

conv2d_1 (Conv2D)            (None, 112, 112, 16)      1168

max_pooling2d_1 (MaxPooling  (None, 56, 56, 16)        0
2D)

batch_normalization_2 (Batc  (None, 56, 56, 16)        64
hNormalization)

dropout (Dropout)            (None, 56, 56, 16)        0

conv2d_2 (Conv2D)            (None, 56, 56, 128)       18560

max_pooling2d_2 (MaxPooling  (None, 28, 28, 128)       0
2D)

batch_normalization_3 (Batc  (None, 28, 28, 128)       512
hNormalization)

dropout_1 (Dropout)          (None, 28, 28, 128)       0

global_average_pooling2d (G  (None, 128)               0
lobalAveragePooling2D)

dense (Dense)                (None, 1)                 129

=================================================================
Total params: 20,701
Trainable params: 20,391
Non-trainable params: 310
```

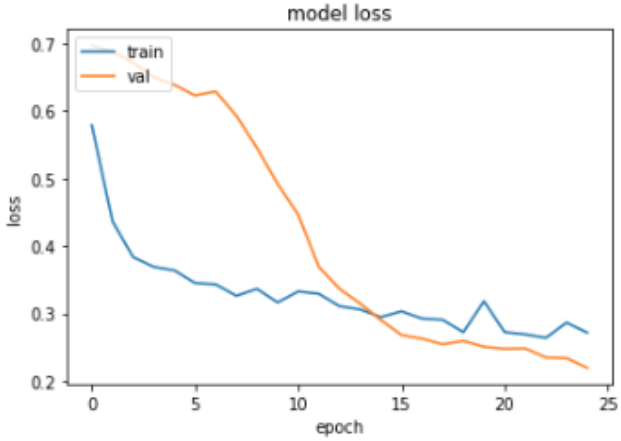
شکل ۱۳- معماری مدل

سپس با آموزش مدل با ۲۵ مرحله به دقت مورد نظر رسیدم.

نمودارهای دقت و خطا در شکل ۱۴ و ۱۵ قابل مشاهده می باشد.



شکل ۱۴- نمودار دقت مدل روی داده‌های **train** و **validation**



شکل ۱۵- نمودار **loss** مدل روی داده‌های **train** و **validation**

پ.

برای تشخیص زنده بودن بر اساس پلک زدن به مراحل زیر نیاز می باشد:

۱. در مرحله اول ما باید یک کلسیفایر داشته باشیم که بتواند در real-time صورت و چشم را تشخیص دهد. و لوکیشن آن هارا در هر فریم به ما بدهد.

۲. در گام بعد نیازمند مدلی می باشیم که بتواند چشم باز و بسته را تشخیص دهد. به دلیل اینکه در هر پلک زدن چشم یکبار بسته و باز می شود. و می‌توانیم شرطی داشته باشیم که اگر چشم مثلاً برای ۴ فریم بسته بود و بعد باز شد این پلک زدن محسوب شود.

۳. حالا که مدل تشخیص بسته شدن پلک و همچنین تشخیص دهنده صورت و چشم هارا داریم نیاز به مدلی داریم که بتواند صورت تشخیص داده شده را انکود کند و به بردار تبدیل کند به صورتی که بتوانیم صورت دو فرد را از هم تشخیص دهیم.

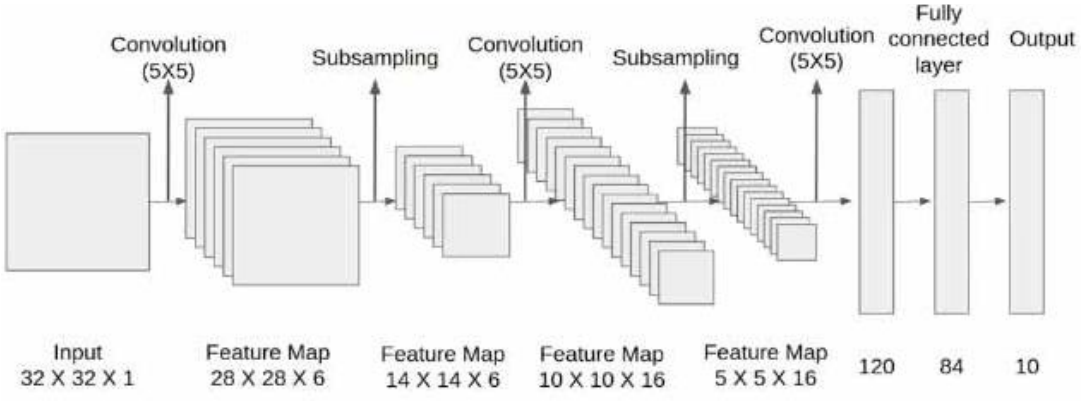
ت.

LeNet-5: قبل از ظهور معماری LeNet-5 شناسایی کاراکتر با استفاده از مهندسی ویژگی دستی و مدل‌های یادگیری ماشین برای یادگیری طبقه‌بندی این ویژگی‌ها انجام می‌شد. معماری LeNet-5 کار مهندسی ویژگی دستی را حذف کرد؛ زیرا خود شبکه به‌طور خودکار بهترین ویژگی‌ها را از تصاویر خام ورودی استخراج می‌کند. این شبکه در تشخیص دست نوشته ها و کلاسبندی آن ها کاربرد زیادی دارد. از ویژگی های اصلی این شبکه می توان به موارد زیر اشاره کرد:

-هر لایه‌ی کانولوشنی از سه بخش تشکیل شده: کانولوشن، pooling و تابع فعالسازی غیرخطی

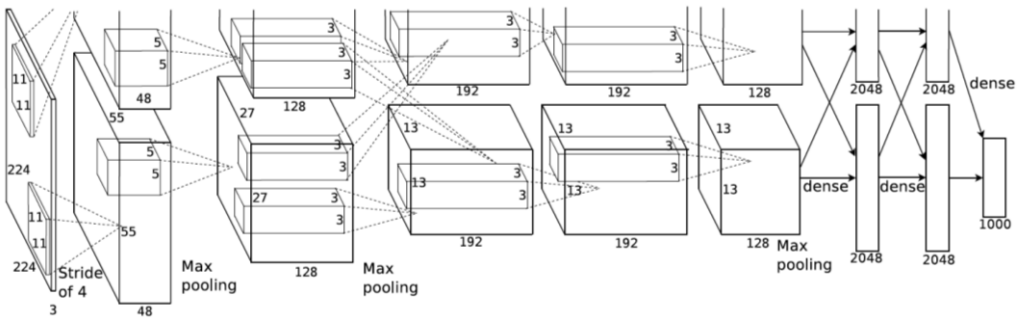
-از کانولوشن برای استخراج ویژگی‌های مکانی استفاده شده است

-لایه‌ی average pooling که به نوعی عملیات Subsampling را انجام می‌دهد



شکل ۱۶- معماری **LeNet-5**

AlexNet: این معماری شامل ۵ لایه کانولوشن ادغام شده با لایه های max-pooling و سه لایه تماما متصل است. چیزی که این مدل را متفاوت می‌کند، سرعت انجام وظیفه و استفاده از «GPU» برای یادگیری است. در دهه ۸۰ میلادی، برای یادگیری یک شبکه عصبی از «CPU» استفاده می‌کردند ولی AlexNet با استفاده از «GPU» سرعت این یادگیری را ده برابر کرد.



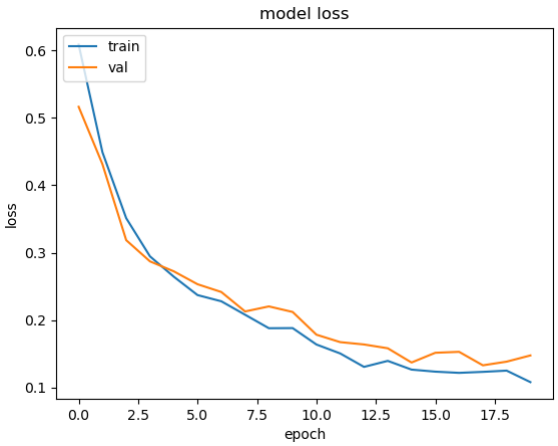
شکل ۱۷- معماری **AlexNet**

بنظر من استفاده از LeNet-5 بهتر است و دلیل اصلی اون استفاده از Average pooling می باشد.

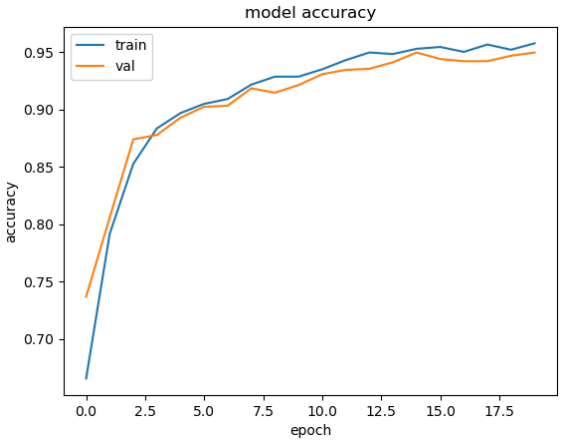
ث.

مراحل گفته شده در قسمت قبل پیاده سازی شد و برای مدل تشخیص چشم باز و بسته از LeNet استفاده شد.

نتیجه آموزش این مدل و نمودار دقت و خطا را در ۲۰ مرحله مشاهده می کنید.



شکل ۱۸- نمودار **loss** مدل طی ۲۰ ایپاک



شکل ۱۹- نمودار دقت مدل طی ۲۰ ایپاک

در ادامه ابتدا عکسی از صورت دو شخص حاضر در دوربین را برای انکود کردن وارد کردم و سپس برنامه شروع به کار کرد. در هر فریم از فیلم چشم های تشخیص داده شده برای کلاس‌بندی بین باز و بسته به مدل داده شد و نتیجه مشخص شد سپس با توجه به انکود صورت شناخته شده و مقایسه با انکودینگ های اولیه هر صورت به فرد مورد نظر نسبت داده شد سپس بعد از بسته شدن چشم هر کس و باز شدن دوباره آن الگوریتم آن را پلک زدن تشخیص می دهد و نام آن فرد را بالای صورتش درج می کند.

فیلم خروجی برنامه در فولدر سوال دوم موجود می باشد.

پاسخ ۳ . تشخیص کاراکتر نوری (Optical character recognition)

الف.

می‌توان گفت تقریباً یکسان هستند و تفاوت بین شبکه های عصبی کانولوشنال و شبکه های کانولشنی عمیق در تفاوت تعداد لایه های دو شبکه می‌باشد و معمولا شبکه‌های کانولوشنی عمیق بین ۳۰ تا ۱۰۰ لایه عمق دارند.

ب.

momentum: ممنتوم با به منظور کاهش واریانس در گرادیان کاهشی تصادفی استفاده می شود. انگار که از تنزل گرادیان نه به عنوان سرعت بلکه به عنوان شتاب استفاده می شود. ممنتوم همگرایی به جهت مورد نظر را تسریع بخشیده و از گرایش به جهات نامربوط پیشگیری می‌کند. ممنتوم گرادیان مراحل گذشته را در حافظه خود نگه می دارد و از آن برای تعیین سرعت گرادیان استفاده می کند. (کمک به فرار از بهینه های محلی و عبور سریعتر از یال ها)

Adam: نرخ یادگیری در الگوریتم بهینه سازی آدام برای هر یک از وزن‌های شبکه (پارامترها) حفظ می‌شود و این نرخ با شروع فرآیند یادگیری به صورت جداگانه تطبیق داده می‌شود.در این روش ، هر یک از نرخ‌های یادگیری برای پارامترهای مختلف از گشتاورهای اول و دوم گرادیان‌ها محاسبه می‌شوند.در آدام به جای انطباق نرخ‌های یادگیری پارامترها تنها بر اساس میانگین گشتاور اول (یعنی Mean) از میانگین گشتاور دوم گرادیان‌ها (واریانس غیر مرکزی) هم استفاده می‌شود.

Adadelta: بهینه سازی آدادلتا یک روش نزولی گرادیان تصادفی است که بر اساس نرخ یادگیری تطبیقی(adaptive) که مشکل اساسی را حل می کند:

۱.کاهش مداوم نرخ یادگیری در طول آموزش.

۲.نیاز به انتخاب دستی نرخ یادگیری کلی

Adadelta یک توسعه قوی تر از Adagrad است که نرخ یادگیری را بر اساس یک پنجره متحرک از به روز رسانی گرادیان تطبیق می دهد. به این ترتیب، Adadelta حتی زمانی که به روز رسانی های زیادی انجام شده است، به یادگیری ادامه می دهد.

ج.

ابتدا تصاویر مجموعه داده HODA برای شناسایی و طبقه بندی به سیستم DCNN وارد می شوند.

ابعاد تصاویر مجموعه داده HODA متفاوت است. همه تصاویر باید سازگار باشند و سایز و شکل یکسان داشته داشته باشند.

مجموعه داده ابتدا با اعمال یک آستانه میانه به یک ماتریس باینری تبدیل شد. علاوه بر این،عملیات morphological، برای صاف کردن لبه و کاهش نویز استفاده شد. سپس داده‌ها به سایز ۴۰ در ۴۰ برده شد. در نهایت برای تبدیل داده‌های کتگوریکال از one hot encoding استفاده شد.

```
from HodaDatasetReader import read_hoda_cdb,read_hoda_dataset

print('Reading train dataset (Train 60000.cdb)...')
X_train, Y_train = read_hoda_dataset(dataset_path='drive/My Drive/Extra/Q3/Train 60000.cdb',images_height=40,images_width=40,one_hot=True,reshape=True)
print('Reading test dataset (Test 20000.cdb)...')
X_test, Y_test = read_hoda_dataset(dataset_path='drive/My Drive/Extra/Q3/Test 20000.cdb',images_height=40,images_width=40,one_hot=True,reshape=True)
print('Reading remaining samples dataset (RemainingSamples.cdb)...')
X_remaining, Y_remaining = read_hoda_dataset('drive/My Drive/Extra/Q3/RemainingSamples.cdb',images_height=40,images_width=40,one_hot=True,reshape=True)
```

```
Reading train dataset (Train 60000.cdb)...
Reading test dataset (Test 20000.cdb)...
Reading remaining samples dataset (RemainingSamples.cdb)...
```

```
# reshape input digit images to 64x64x1
X_train = X_train.reshape([-1, 40, 40, 1])


# reshape input letter images to 64x64x1
X_test = X_test.reshape([-1, 40, 40, 1])
```

شکل ۲۰- خواندن داده‌ها و انجام پیش‌پردازش‌ها

معماری مدل طبق طراحی گفته شده در مقاله انجام شد.

در صفحه بعد summary و شکل معماری مدل را مشاهده می کنید.

مدل ۲۸ لایه دارد .

-تصاویر در ابتدا در اولین لایه که به عنوان لایه ورودی شناخته می شود ذخیره می شوند. این لایه ابعاد (ارتفاع، عرض) داده های ورودی و تعداد کانال ها (اطلاعات RGB) را تعریف می کند.

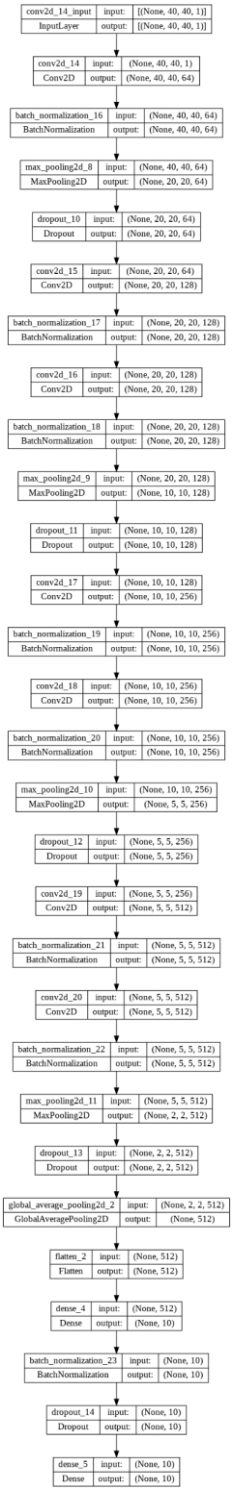
-لایه کانولوشن از چندین فیلتر برای استخراج ویژگی های اساسی یک تصویر استفاده می کند. و حاصل ضرب نقطه ای نورون ورودی m*m و فیلتر n*n که روی ورودی convolve شده است را محاسبه می کند. با اضافه کردن هر لایه کانولوشنی، CNN پیچیدگی خود را افزایش می دهد و قسمتهای بیشتری از تصویر را مشخص می کند. لایه های قبلی بر ویژگی های ساده مانند رنگ ها و لبه ها تمرکز می کنند. همانطور که داده های تصویر در لایه های CNN پیش می روند ، عناصر یا اشکال بزرگتر شیء را تشخیص می دهند تا در نهایت شی مورد نظر را شناسایی کنند.

- لایه Pooling به عنوان لایه "Sub-Sampling" نیز شناخته می شود. برای کاهش ابعاد تصویر بین دو لایه کانولوشنی قرار می گیرد. علاوه بر این، Pooling عملیات پیچیدگی محاسباتی شبکه را به حداقل می رساند.

-برای جلوگیری از overfitting از dropout استفاده شده است. یعنی اینکه در هر مرحله از آموزش، نودهایی از شبکه، با احتمال p-۱ کنار گذاشته شده و نودهای دیگری با احتمال p، حفظ می‌شوند. بنابراین یک شبکه کاهش یافته باقی می‌ماند.

-معمولا آخرین لایه های یک شبکه عصبی کانولوشن برای طبقه بندی را لایه های fully connected تشکیل می‌دهند. یکی از کاربردهای اصلی این لایه در شبکه کانولوشن، استفاده به عنوان طبقه بند یا کلاسیفایر (Classifier) است. یعنی مجموعه ویژگی‌های استخراج شده با استفاده از لایه های کانولوشنی درنهایت تبدیل به یک بردار می‌شوند. درنهایت این بردار ویژگی به یک کلاسیفایر داده می‌شود تا کلاس درست را شناسایی کند.

لایه خروجی هم یک لایه dense می‌باشد با ۱۰ نورون خروجی به ازای تمام کلاس‌های خروجی.



شکل ۲۱- معماری مدل مقاله

د.

هر سه مدل را با ۲۰ epoch دادم که زمان آموزش طولانی نشود و هم نتیجه قابل مقایسه باشد.

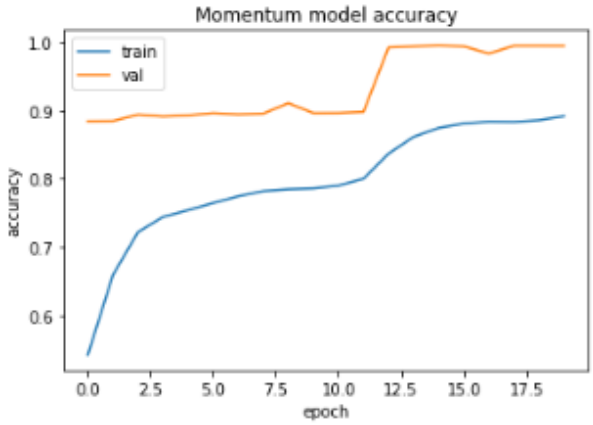
روش اول : بهینه سازی با Momentum

```
epochs = 20
batch_size = 32

opt= tensorflow.keras.optimizers.SGD(learning_rate=0.001,  momentum=0.9)
curr_model = create_model(opt)
history=curr_model.fit(X_train, Y_train,
                    validation_data=(X_test, Y_test),
                    epochs=epochs, batch_size=batch_size, verbose=1)
print("================================================================================")
```

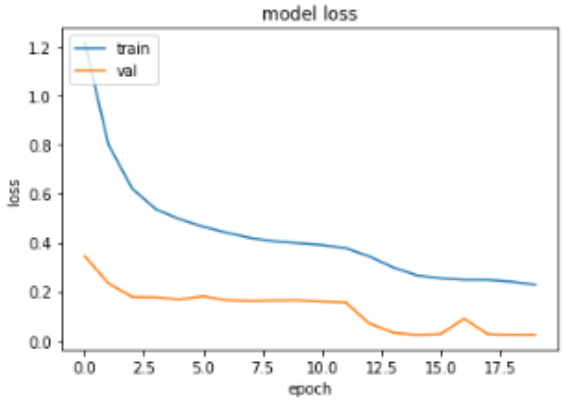
```
Epoch 1/20
1875/1875 [==============================] - 42s 22ms/step - loss: 1.2173 - accuracy: 0.5423 - val_loss: 0.3446 - val_accuracy: 0.8844
Epoch 2/20
1875/1875 [==============================] - 39s 21ms/step - loss: 0.8002 - accuracy: 0.6589 - val_loss: 0.2336 - val_accuracy: 0.8846
Epoch 3/20
1875/1875 [==============================] - 39s 21ms/step - loss: 0.6207 - accuracy: 0.7219 - val_loss: 0.1781 - val_accuracy: 0.8939
Epoch 4/20
1875/1875 [==============================] - 39s 21ms/step - loss: 0.5367 - accuracy: 0.7441 - val_loss: 0.1769 - val_accuracy: 0.8913
Epoch 5/20
1875/1875 [==============================] - 40s 22ms/step - loss: 0.4960 - accuracy: 0.7540 - val_loss: 0.1683 - val_accuracy: 0.8929
Epoch 6/20
1875/1875 [==============================] - 40s 22ms/step - loss: 0.4650 - accuracy: 0.7645 - val_loss: 0.1813 - val_accuracy: 0.8963
Epoch 7/20
```

شکل ۲۲- آموزش مدل با بهینه‌ساز **momentum**

در بهترین نتیجه پارامتر مومنتوم را ۰.۹ و نرخ آموزش را ۰.۰۰۱ قرار دادم. نمودار خطا و دقت مدل با این بهینه‌ساز را در شکل ۲۳ و ۲۴ مشاهده می کنید.



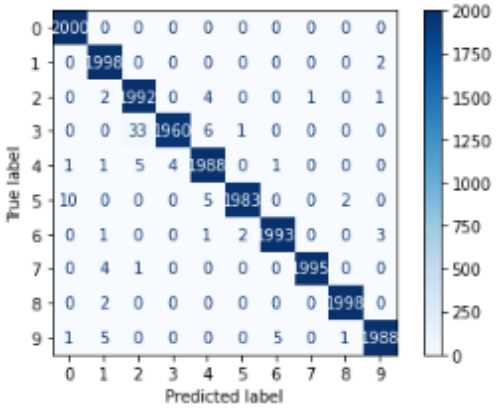
شکل ۲۳- نمودار دقت مدل با بهینه‌ساز **momentum**



شکل ۲۴- نمودار **loss** مدل با بهینه‌ساز **momentum**

	precision	recall	f1-score	support
0	0.99	1.00	1.00	2000
1	0.99	1.00	1.00	2000
2	0.98	1.00	0.99	2000
3	1.00	0.98	0.99	2000
4	0.99	0.99	0.99	2000
5	1.00	0.99	0.99	2000
6	1.00	1.00	1.00	2000
7	1.00	1.00	1.00	2000
8	1.00	1.00	1.00	2000
9	1.00	0.99	1.00	2000
accuracy			0.99	20000
macro avg	0.99	0.99	0.99	20000
weighted avg	0.99	0.99	0.99	20000

شکل ۲۵- نتیجه‌ی مدل با بهینه‌ساز **momentum**



شکل ۲۶- ماتریس آشفتگی مدل با بهینه‌ساز **momentum**

همانطور که مشاهده کردید مدل به دقت قابل قبولی رسید و از بیش برازش دوری کرد و نتیجه دقت و خطای داده ارزیابی بسیار قابل قبول می باشد.

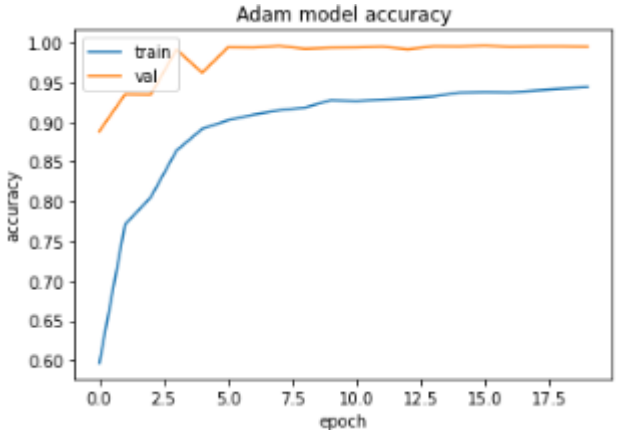
روش دوم : بهینه سازی با Adam

```
epochs = 20
batch_size = 32
opt=tensorflow.keras.optimizers.Adam()
curr_model =  create_model(opt)
history=curr_model.fit(X_train, Y_train,
                   validation_data=(X_test, Y_test),
                   epochs=epochs, batch_size=batch_size, verbose=1)
print("================================================================================")
```

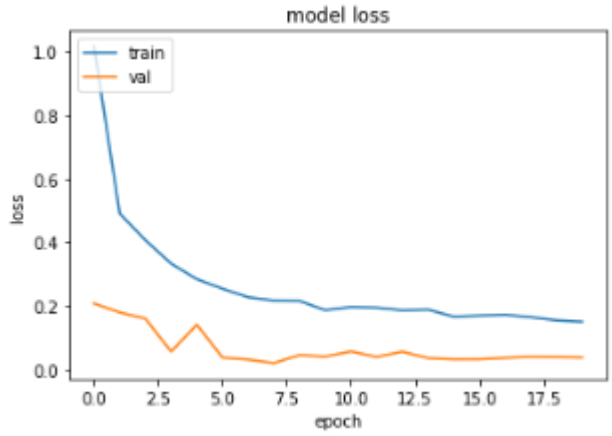
```
Epoch 1/20
1875/1875 [==============================] - 46s 23ms/step - loss: 1.0154 - accuracy: 0.5970 - val_loss: 0.2076 - val_accuracy: 0.8882
Epoch 2/20
1875/1875 [==============================] - 42s 22ms/step - loss: 0.4902 - accuracy: 0.7710 - val_loss: 0.1794 - val_accuracy: 0.9344
Epoch 3/20
1875/1875 [==============================] - 40s 22ms/step - loss: 0.4066 - accuracy: 0.8051 - val_loss: 0.1597 - val_accuracy: 0.9341
Epoch 4/20
1875/1875 [==============================] - 40s 22ms/step - loss: 0.3330 - accuracy: 0.8638 - val_loss: 0.0563 - val_accuracy: 0.9909
Epoch 5/20
1875/1875 [==============================] - 42s 22ms/step - loss: 0.2841 - accuracy: 0.8911 - val_loss: 0.1410 - val_accuracy: 0.9616
Epoch 6/20
1875/1875 [==============================] - 40s 21ms/step - loss: 0.2539 - accuracy: 0.9019 - val_loss: 0.0375 - val_accuracy: 0.9935
Epoch 7/20
1875/1875 [==============================] - 40s 21ms/step - loss: 0.2272 - accuracy: 0.9090 - val_loss: 0.0311 - val_accuracy: 0.9933
Epoch 8/20
1875/1875 [==============================] - 41s 22ms/step - loss: 0.2170 - accuracy: 0.9144 - val_loss: 0.0203 - val_accuracy: 0.9954
Epoch 9/20
1875/1875 [==============================] - 40s 21ms/step - loss: 0.2153 - accuracy: 0.9179 - val_loss: 0.0453 - val_accuracy: 0.9916
Epoch 10/20
```

شکل ۲۷- آموزش مدل با بهینه‌ساز **Adam**

نمودار خطا و دقت مدل با این بهینه‌ساز را در شکل ۲۳ و ۲۴ مشاهده می کنید.



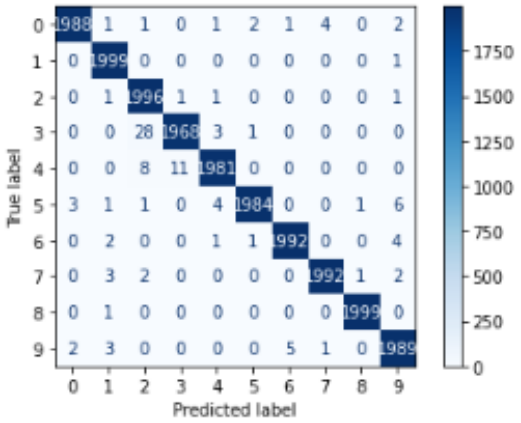
شکل ۲۸- نمودار دقت مدل با بهینه‌ساز **Adam**



شکل ۲۹- نمودار **loss** مدل با بهینه‌ساز **Adam**

Adam Result:	precision	recall	f1-score	support
0	1.00	0.99	1.00	2000
1	0.99	1.00	1.00	2000
2	0.98	1.00	0.99	2000
3	0.99	0.98	0.99	2000
4	0.99	0.99	0.99	2000
5	1.00	0.99	0.99	2000
6	1.00	1.00	1.00	2000
7	1.00	1.00	1.00	2000
8	1.00	1.00	1.00	2000
9	0.99	0.99	0.99	2000
accuracy			0.99	20000
macro avg	0.99	0.99	0.99	20000
weighted avg	0.99	0.99	0.99	20000

شکل ۳۰- نتیجه‌ی مدل با بهینه‌ساز **Adam**



شکل ۳۱- ماتریس آشفتگی مدل با بهینه‌ساز **Adam**

همانطور که مشاهده کردید مدل با ۲۰ مرحله آموزش به دقت ارزیابی ۹۹ و دقت آموزش۹۱ رسید.

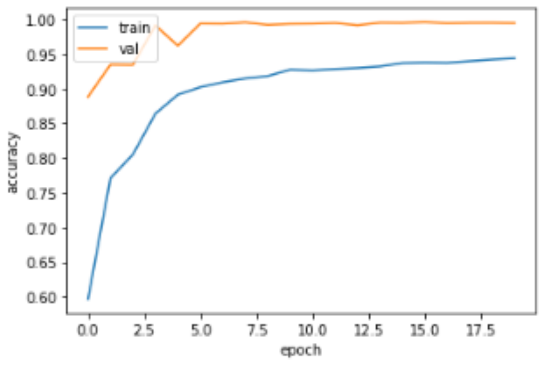
روش سوم: بهینه سازی با AdaDelta

```
epochs = 20
batch_size = 32
opt=tensorflow.keras.optimizers.Adadelta()
curr_model =  create_model(opt)
history=curr_model.fit(X_train, Y_train,
                  validation_data=(X_test, Y_test),
                  epochs=epochs, batch_size=batch_size, verbose=1)
print("================================================================================")
```

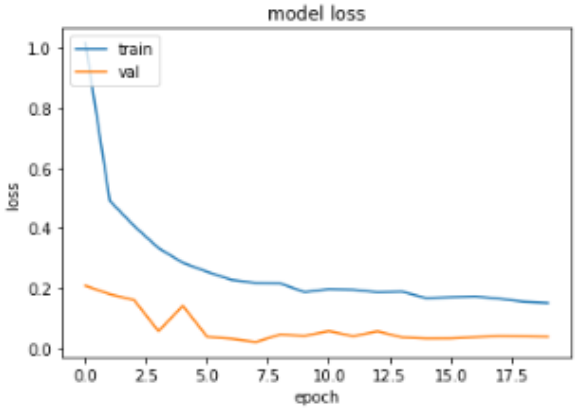
```
Epoch 1/20
1875/1875 [==============================] - 44s 23ms/step - loss: 2.3332 - accuracy: 0.1461 - val_loss: 1.9971 - val_accuracy: 0.3348
Epoch 2/20
1875/1875 [==============================] - 42s 23ms/step - loss: 2.1517 - accuracy: 0.2064 - val_loss: 1.8782 - val_accuracy: 0.3038
Epoch 3/20
1875/1875 [==============================] - 41s 22ms/step - loss: 1.9133 - accuracy: 0.2995 - val_loss: 1.6981 - val_accuracy: 0.4448
Epoch 4/20
1875/1875 [==============================] - 41s 22ms/step - loss: 1.7874 - accuracy: 0.3519 - val_loss: 1.5849 - val_accuracy: 0.4910
Epoch 5/20
1875/1875 [==============================] - 43s 23ms/step - loss: 1.7267 - accuracy: 0.3805 - val_loss: 1.5398 - val_accuracy: 0.5282
Epoch 6/20
1875/1875 [==============================] - 41s 22ms/step - loss: 1.6710 - accuracy: 0.4114 - val_loss: 1.4990 - val_accuracy: 0.5426
```

شکل ۳۲- آموزش مدل با بهینه‌ساز **AdaDelta**

نمودار خطا و دقت مدل با این بهینه‌ساز را در شکل ۳۳ و ۳۴ مشاهده می کنید.



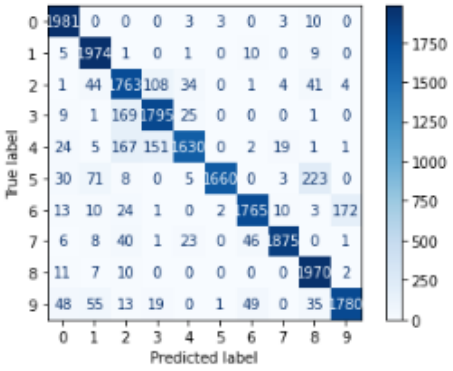
شکل ۳۳- نمودار دقت مدل با بهینه‌ساز **AdaDelta**



شکل ۳۴- نمودار **loss** مدل با بهینه‌ساز **AdaDelta**

	precision	recall	f1-score	support
0	0.93	0.99	0.96	2000
1	0.91	0.99	0.95	2000
2	0.80	0.88	0.84	2000
3	0.87	0.90	0.88	2000
4	0.95	0.81	0.88	2000
5	1.00	0.83	0.91	2000
6	0.94	0.88	0.91	2000
7	0.98	0.94	0.96	2000
8	0.86	0.98	0.92	2000
9	0.91	0.89	0.90	2000
accuracy			0.91	20000
macro avg	0.91	0.91	0.91	20000
weighted avg	0.91	0.91	0.91	20000

شکل ۳۵- نتیجه‌ی مدل با بهینه‌ساز **AdaDelta**



شکل ۳۶- ماتریس آشفتگی مدل با بهینه‌ساز **AdaDelta**

همانطور که مشاهده کردید مدل با ۲۰ مرحله آموزش به دقت مورد نظر نرسید و نسبت به دو روش دیگر عملکرد ضعیف تری داشت.

مقایسه نتیجه سه روش بعد از ۲۰ مرحله آموزش را در جدول ۱ مشاهده می‌کنید.

جدول ۱- مقایسه‌ی ۳ بهینه‌ساز

Optimizers	Acc-train	Acc-val	Loss-train	Loss-val
Momentum	۰.۸۹۱۷	۰.۹۹۴۸	۰.۲۲۸۶	۰.۰۲۴۴
Adam	۰.۹۴۳۶	۰.۹۹۴۴	۰.۱۵۰۲	۰.۰۳۷۶
AdaDelta	۰.۵۳۶۳	۰.۷۳۰۱	۱.۳۸۲۵	۱.۰۴۴۱

دو روش ممنتوم و Adam نتایج شبیه به هم و خوبی داشتند در حالیکه مدل با بهینه‌ساز AdaDelta عملکرد بسیار ضعیفتری در مقایسه با دو بهینه‌ساز دیگر داشت.

.۵

معماری مدل همان معماری که در قسمت ج توضیح دادیم بود اما Batch-size=32 و learning-rate=0.01 و momentum=0.9 پارامتر های مشخص شده بود.