

به نام خدا

تمرین پنجم درس هوش مصنوعی

محمد امین سلطانی چم حیدری

۸۱۰۶۰۱۰۸۱

استاد مربوطه:

دکتر مسعود شریعت پناهی

۱۴۰۲ بهار

## **بخش اول: مفاهیم پایه**

الف) برای هریک از کاربردهای دسته‌بندی (classification) و پیش‌بینی (regression) دو نمونه تابع زیان loss معرفی کرده و عملکرد آن‌ها را بصورت کوتاه توضیح دهید.

## regression

یکی از معروف‌ترین و معمول‌ترین توابع زیان در تحلیل رگرسیونی، میانگین مربعات خطای (Means Square Error) است که به اختصار MSE نامیده می‌شود. این تابع زیان، میانگین مربعات فاصله بین مقدار پیش‌بینی و واقعی را محاسبه می‌کند. با توجه به استفاده از توان ۲ در محاسبه MSE، شکل این تابع زیان بر حسب مقدارهای پیش‌بینی (یا خط) به صورت سهمی خواهد بود.

یکی دیگر از توابع زیان، میانگین قدرمطلق خطای (Mean Absolute Error) است که به اختصار MAE نیز نامیده می‌شود. این تابع زیان، به مانند MSE از فاصله بین مقدار پیش‌بینی و واقعی به عنوان معیار استفاده کرده ولی جهت این تفاضل را در نظر نمی‌گیرد. بنابراین در محاسبه خطای MAE فقط میزان فاصله و نه جهت فاصله به کار می‌رود.

## classification

تابع زیان طبقه‌بندی دو دسته‌ای (binary classification) برای مسائل طبقه‌بندی دو دسته‌ای binary cross entropy استفاده می‌شود. این تابع خطای تفاضل بین احتمال پیش‌بینی شده توسط شبکه برای دسته‌ی مثبت و دسته‌ی منفی و احتمالات واقعی برچسب‌های داده‌ها است. با استفاده از این تابع خطای شبکه عصی به شیوه‌ای که بتواند احتمال اینکه یک داده برای دسته‌ی مثبت باشد یا نباشد را حساب کند، آموزش می‌یابد.

تابع زیان لگاریتمی (Logarithmic Loss) یکی از معروف‌ترین توابع زیانی است که برای مسائل دسته‌بندی دودویی (binary classification) و چند دسته‌ای (multi-class classification) استفاده می‌شود. این تابع، میزان تفاوت بین پیش‌بینی مدل و برچسب‌های واقعی داده‌ها را اندازه‌گیری می‌کند.

هدف این تابع، کمینه کردن میزان اختلاف بین پیش‌بینی مدل و برچسب‌های واقعی داده‌های است. همچنین، می‌توان گفت که این تابع، یک معیار خوب برای ارزیابی عملکرد مدل است، به این دلیل که به صورت مستقیم با احتمالات پیش‌بینی شده توسط مدل کار می‌کند و مقدار خروجی آن، بین ۰ و ۱ است.

ب) عملکرد لایه Batch Normalization را در شبکه های عمیق توضیح دهید و اهمیت این لایه را در شبکه های عصبی عمیق را توضیح دهید.

یکی از روش های مهم و موثر برای بهبود عملکرد شبکه های عصبی است. این روش در هر لایه از شبکه، ورودی های لایه را به صورت مستقل و با میانگین صفر و واریانس یک، نرمالیز می کند. ابتدا ورودی نرمال می شود. سپس این ورودی وارد شده تغییر مقیاس داده می شود (scaling) و در نهایت ورودی ها جابجا می شوند (offsetting).

عملکرد Batch Normalization به دو صورت مختلف است:

۱. کنترل مشکل بروزگرادیان محو شونده (Vanishing Gradient) و شیب گیری بیش از حد (Exploding Gradient): با نرمال کردن ورودی های لایه، ورودی های لایه برای لایه های قبلی بهبود یافته و در نتیجه مشکل بروزگرادیان محو شونده و یا شیب گیری بیش از حد کاهش می یابد. با توجه به اینکه در شبکه های عمیق این موضوع در اثر تعداد زیاد لایه ها زیاد بوجود می آید این عملکرد Batch Normalization در شبکه های عصبی عمیق بسیار مهم می باشد.

۲. کاهش زمان آموزش: با نرمال کردن ورودی های لایه، توزیع ورودی های لایه ثابت می شود و تغییرات در ورودی های لایه در یادگیری پارامترها مهم تر می شود، در نتیجه زمان آموزش شبکه کاهش می یابد.

## مزایای نرمال سازی دسته ای

- بهبود عملکرد شبکه در مسائل دسته بندی و تشخیص شیء
- افزایش سرعت آموزش شبکه و کاهش نرخ بروزگرادیان محو شونده و شیب گیری بیش از حد
- کمک به جلوگیری از بیش برازش شبکه
- با activation normalization لایه پنهان، نرمال سازی دسته ای روند آموزش را سرعت می بخشد.
- یکنواخت کردن (smoothens) تابع زیان: تابع زیان را یکنواخت می کند که به نوبه خود با بهینه سازی پارامترهای مدل سرعت آموزش مدل را بهبود می بخشد.

پ) در مورد عملکرد لایه‌های Batch Normalization و Layer Normalization و تفاوت آن‌ها با لایه

Normalization توضیح دهید. (برای آشنایی با عملکرد این لایه‌ها می‌توانید از این [لینک](#) کمک بگیرید.)

## Layer Normalization

از طرحی مشابه با Batch Norm استفاده می‌کند، با این حال، نرمال سازی در هر بعد اعمال نمی‌شود، بلکه در هر نقطه داده اعمال می‌شود. به عبارت دیگر، با Layer Norm، هر نقطه داده را به طور جداگانه normalize می‌کنیم. علاوه بر این، میانگین و واریانس هر نقطه داده در تمام واحدهای پنهان (یعنی نورون‌ها) لایه به اشتراک گذاشته می‌شود. به عنوان مثال، در پردازش تصویر، ما هر تصویر را مستقل از هر تصویر دیگری normalize می‌کنیم، میانگین و واریانس هر تصویر بر روی تمام پیکسل‌ها، کanal‌ها و نورون‌های لایه محاسبه می‌شود. با استفاده از این انحراف میانگین و استاندارد، مراحل بعدی مانند Batch Norm است که در آن مقدار ورودی کاهش می‌یابد، سپس بر انحراف استاندارد تقسیم می‌شود، و سپس با ضرایب یادگرفته شده تبدیل می‌شود.

برای تعریف instance normalization لازم است ابتدا Group normalization تعریف شود.

Norm Instance نیز اصلاح Norm Batch است با تنها تفاوت این است که میانگین و واریانس بر روی بعد دسته‌ای محاسبه نمی‌شود. به عبارت دیگر، تنها پیکسل‌های موجود در یک تصویر و کanal مشابه، میانگین و واریانس نرمال سازی را به اشتراک می‌گذارند. با چنین محدوده نرمال سازی محدود، به نظر می‌رسد زمانی که ورودی داده‌های تصویری باشد، استفاده از Norm Instance در computer vision محدود باشد.

## Group Normalization

یک معامله بین Norm Group و Norm Layer است. باید به این نکته توجه کرد که تفاوت عمده بین Instance Norm و Layer Norm در این است که Instance Norm بعد کanal را به محاسبات می‌برد در حالی که Group Norm این کار را نمی‌کند. از طرف دیگر Group Norm کanal‌ها را به گروه‌ها گروه بندی می‌کند و در داخل هر گروه normalize می‌شود.

پیش پردازش داده ها

## بخش دوم: تشخیص (دسته‌بندی) تابلوهای راهنمایی و رانندگی

یکی از کاربردهای مهم شبکه‌های عصبی، کمک به سامانه رانندگی خودکار (auto pilot) در خودروهای خودران<sup>۱</sup> از طریق تشخیص تابلوهای راهنمایی و رانندگی است که سامانه را قادر می‌سازد تا در رویارویی با تابلوهای مختلف، واکنش مناسبی از خود نشان دهد. در این بخش می‌خواهیم با طراحی دو نوع شبکه عصبی و تربیت آن‌ها به کمک دادگان ترافیکی موجود، تابلوهای راهنمایی و رانندگی را تشخیص دهیم.

ابتدا دادگان مورد نظر [German Traffic Sign Recognition Benchmark (GTSRB)] را از این [لینک](#) دانلود کنید. سپس یک بار با استفاده از یک شبکه MLP و بار دیگر با استفاده از یک شبکه پیچشی (CNN) و با بهره گیری از کتابخانه‌های مختلف این داده‌ها را در ۴۳ گروه دسته بندی کنید

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import pylab as pl
from sklearn import preprocessing
import cv2
from sklearn.model_selection import train_test_split
from tensorflow import keras
import tensorflow as tf
from tensorflow.keras.models import load_model
import netron
from sklearn.metrics import confusion_matrix
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
```

ابتدا کتابخانه‌های مورد نیاز برای فراخوانی می‌شوند. از آنجایی که امکان دارد فراخوانی بعضی کتابخانه‌ها فراموش شده باشد و همچنین چون برخی کدها از سایت‌های موجود در منبع کپی شده‌اند در ابتدای اجرای برخی قسمت‌ها امکان دارد این کتابخانه‌ها مجدداً فراخوانی شوند که در روند تربیت مشکلی بوجود نمی‌آید.

← نمایش data ← matplotlib و رسم نمودار

← کار با داده‌های ساختار یافته مانند جداول، فایل CSV و Excel و ...

← انجام عملیات ریاضی، آماری و عددی ← Numpy

← Sickit\_learn یادگیری ماشین و تحلیل داده  
 ← Pylab تجسم داده های علمی و عملی ترکیبی از numpy و matplotlib می باشد.  
 ← Tensorflow تربیت مدل شبکه عصبی  
 ← Dense تعریف تعداد لایه ها و نورون های شبکه  
 ← SGD بهینه ساز شبکه از نوع کاهش گرادیانی.  
 ← CV2 فراخوانی تصاویر از روی آدرس های موجود در فایل csv  
 ← Load model ذخیره plot از مدل شبکه عصبی  
 ← Netron نمایش مدل ذخیره شده توسط load model  
 ← Train test split تقسیم داده ها به دادهای train , validation  
 ← Conv2D اضافه کردن لایه های پیچشی به شبکه پیچشی  
 ← Maxpooling2D,Averagepooling2D اضافه کردن لایه های pooling به شبکه عصبی

```

Train_df = pd.read_csv("Train.csv")
Test_df = pd.read_csv("Test.csv")
Train_df.head()
  
```

	Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId	Path
0	27	26	5	5	22	20	20	Train/20/00020_00000_00000.png
1	28	27	5	6	23	22	20	Train/20/00020_00000_00001.png
2	29	26	6	5	24	21	20	Train/20/00020_00000_00002.png
3	28	27	5	6	23	22	20	Train/20/00020_00000_00003.png
4	28	26	5	5	23	21	20	Train/20/00020_00000_00004.png

سپس داده های موجود در فایل CSV را فراخوانی میکنیم که پنج سطر اول آن مطابق شکل بالا نشان داده شده است.

در ادامه با استفاده از مسیرهای موجود در ستون path و با دستور cv2 تصاویر موجود در پوشه ها را برای داده های test , train فراخوانی کرده و برروی آن ها پیش پردازش های مورد نیاز را انجام می دهیم.

## پیش پردازش داده ها:

در این دادگان تصاویر سه کanalه RGB با ابعاد مختلف از تابلوهای راهنمایی و رانندگی موجود است. ابتدا با استفاده از دستور `resize` ابعاد همه تصاویر را  $30 \times 30$  قراردهید. و مقدار هر پیکسل را به ۲۵۵ (بیشترین مقدار ممکن برای هر پیکسل) تقسیم کنید. سپس داده های موجود در پوشش `train` را بصورت تصادفی به نسبت ۰/۹ برای آموزش (training) و ۰/۱ برای ارزیابی (validation) جدا کنید و داده های موجود در پوشش `test` را نیز برای آزمون عملکرد شبکه های خود در نظر بگیرید.

```
import pandas as pd
import cv2

Train_labels1=Train_df["ClassId"].values
Test_labels1=Test_df["ClassId"].values
Train_images1=[]
Test_images1=[]
new_size = (30, 30)

for index, row in Train_df.iterrows():
    img_path = row["Path"]
    train_img = cv2.imread(img_path)
    train_img_rgb = cv2.cvtColor(train_img, cv2.COLOR_BGR2RGB)
    resized_train_img_rgb = cv2.resize(train_img_rgb, new_size)/255
    Train_images1.append(resized_train_img_rgb)
for index, row in Test_df.iterrows():
    img_path = row["Path"]
    test_img = cv2.imread(img_path)
    test_img_rgb = cv2.cvtColor(test_img, cv2.COLOR_BGR2RGB)
    resized_test_img_rgb = cv2.resize(test_img_rgb, new_size)/255
    Test_images1.append(resized_test_img_rgb)

for img in Train_images1:
    print(img.shape)
```

ابتدا داده های ستون `ClassId` را به عنوان خروجی تعریف می کنیم.

سپس دو حلقه `for` یکی برای داده های تست و یکی برای داده های ازمون تعریف می کنیم. در این حلقه با استفاده از `cv2.imread` تصاویر موجود در پوشه ها را فراخوانی میکنیم.

با توجه به اینکه داده های فراخوانی شده با فرمت GBR هستند با استفاده از CV2.COLOR\_BGR2RGB داده های فراخوانی شده را به فرمت RGB تبدیل می کنیم.

در قسمت بعد مطابق خواسته سوال با دستور Resized ابعاد همه داده ها را  $30 * 30$  می کنیم و حاصل را تقسیم بر 255 میکنیم.

دادا های پیش پردازش شده برای قسمت train در شکل زیر نشان داده شده اند:

```
Train_labels1=np.array(Train_labels1)
Test_labels=np.array(Test_labels1)
Train_images1=np.array(Train_images1)
Test_images=np.array(Test_images1)

from sklearn.model_selection import train_test_split
Train_images, Valid_images, Train_labels, Valid_labels = train_test_split(Train_images1, Train_labels1, test_size=0.1)
```

در قسمت بعد همه ی داده های ورودی را به فرمت `array` تبدیل می کنیم و مطابق خواسته سوال داده های `train` را با نسبت 9 به 1 به داده های `validation` تقسیم میکنیم.

حال پس از انجام پیش پردازش های گفته شده داده های ما آماده هستند تا برای آن ها شبکه عصبی و شبکه پیچشی را تعریف کنیم.

پیاده سازی شبکه ها

# شبکه MLP

## الف - شبکه MLP

### معماری شبکه

در طراحی شبکه MLP خود می‌توانید از هر تعداد لایه پنهان و هر تعداد نرون در لایه‌ها استفاده کنید، به شرط آنکه شبکه بدست آمده بیش از حد سنگین و تربیت آن بیش از حد زمان بر و یا دقت نهایی آن از حد مطلوب کمتر نشود. همچنین در صورتی که سرعت آموزش شبکه کم بود (تنها در این بخش) می‌توانید به جای استفاده از همه داده‌های آموزش، از زیرمجموعه‌ای از این داده‌ها استفاده کنید. دقت کنید که در این صورت باید این زیرمجموعه را به صورت تصادفی انتخاب کنید و این زیرمجموعه باید شامل داده‌های تمامی دسته‌ها باشد. در گزارش خود توضیح دهید که در آموزش این شبکه از چه تابع زیانی استفاده کردید و دلیل این تصمیم را نیز مشخصاً بیان کنید. همچنین تابع فعال‌سازی لایه آخر را نیز به دلخواه (و البته با ذکر دلیل) مشخص نمایید.

### آموزش شبکه

برای آموزش شبکه MLP می‌توانید از هر تابع زیان و هر الگوریتم بهینه سازی استفاده کنید. نرخ یادگیری را (در این بخش) با آزمون و خطا و تعیین کنید و توضیح دهید نرخ یادگیری نامناسب چه تاثیری بر روند یادگیری خواهد داشت.

## شبکه MLP

```
# تعریف مدل شبکه MLP
from tensorflow import keras
from tensorflow.keras import layers
from keras.optimizers import SGD
from sklearn.metrics import mean_squared_error
from keras.optimizers import Adam
from tensorflow.keras.models import load_model
import netron

# لایه مروجی
model = keras.Sequential([
    layers.Flatten(input_shape=(30, 30, 3)), # لایه پنهان ۱
    layers.Dense(256, activation='relu'), # لایه پنهان ۲
    layers.Dense(128, activation='relu'), # لایه پنهان ۳
    layers.Dense(43, activation='softmax') # لایه خروجی
])

sgd = SGD(learning_rate=0.001)
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])

history = model.fit(Train_images, Train_labels, epochs=20, validation_data=(Valid_images,Valid_labels))

test_loss, test_acc = model.evaluate(Test_images, Test_labels)
print('Test accuracy:', test_acc)
print('Test loss:', test_loss)

model.save('model.h5')
netron.start('model.h5')
```

برای تعریف شبکه عصبی:

- یک لایه flatten برای تبدیل تصاویر با بردار سه بعدی، به بردارهای یک بعدی تعریف میکنیم
- دو لایه پنهان که یکی 256 و دیگری 128 نورون دارد استفاده می کنیم
- تعداد نورون لایه خروجی را برابر با تعداد خروجی ها در نظر میگیریم
- تابع فعا ساز لایه های پنهان را Relu و لایه خروجی را softmax برای دسته بندی های چند گانه استفاده می شود. این تابع احتمال تمام خروجی ها را در نظر میگیرد و هر خروجی که احتمال آن بیشتر باشد برابر یک و سایر خروجی هارا برابر صفر قرار می دهد و چون مدل ما یک دسته بندی چند گانه است از این تابع برای لایه خروجی استفاده می کنیم.
- تابع بهینه ساز را از نوع SGD یا بهینه ساز با گرادیان کاهشی انتخاب شده است.

- مدل با نرخ های یادگیری  $0.1, 0.05, 0.02, 0.01, 0.005, 0.001$  تربیت شد و نرخ یادگیری برای تربیت نهایی 0.02 در نظر گرفته شد. با توجه به تعداد ایپاک کم اگر نرخ یادگیری خیلی کم در نظر گرفته شود مدل در این تعداد ایپاک به دقت مطلوب نمی رسد و اگر نرخ یادگیری خیلی زیاد در نظر گرفته شود نمودار تابع زیان و خطای در هر ایپاک تغییرات زیادی میکنند و به اصطلاح نمودار smooth نخواهد بود.

- تابع زیان یا **Loss function** را از نوع **categorical cross entropy** تعریف کرده ایم و علت آن نیز این است که در مسائل **classification** از با توجه به این که خروجی ما پیوسته نیست نمی توانیم از توابع زیان **mse** و **mae** (که در مسائل رگرسیون مورد استفاده قرار میگیرند) استفاده کنیم. همچنین چون دسته بندی ما از نوع چندگانه است به جای **categorical cross entropy** از **entropy** استفاده می کنیم.

نکته: با توجه به فیلتر بودن سایت [graphviz.org](http://graphviz.org) و اینکه نتوانستم از سایر منابع کتابخانه **graphviz** را نصب و اجرا کنم برای پلات کردن مدل شبکه از تابع **netron** استفاده شده است.

input

?×30×30×3

InputLayer

Flatten

Dense

**kernel** <2700×256>  
**bias** <256>

ReLU

Dense

**kernel** <256×128>  
**bias** <128>

ReLU

Dense

**kernel** <128×43>  
**bias** <43>

Softmax

dense\_2

MLP مربوط به شبکه Plot

```

train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.plot(train_loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Loss Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.plot(train_acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.title('Accuracy Curve')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
print('Validation Loss:', val_loss[-1])
print('Validation Accuracy:', val_acc[-1])

from sklearn.metrics import confusion_matrix
import seaborn as sns

y_pred = model.predict(Test_images)

y_pred = np.argmax(y_pred, axis=1)
Test_labels=np.argmax(Test_labels, axis=1)
cm = confusion_matrix(Test_labels, y_pred)

plt.figure(figsize=(14,10))
sns.heatmap(cm, annot=True, fmt='d')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

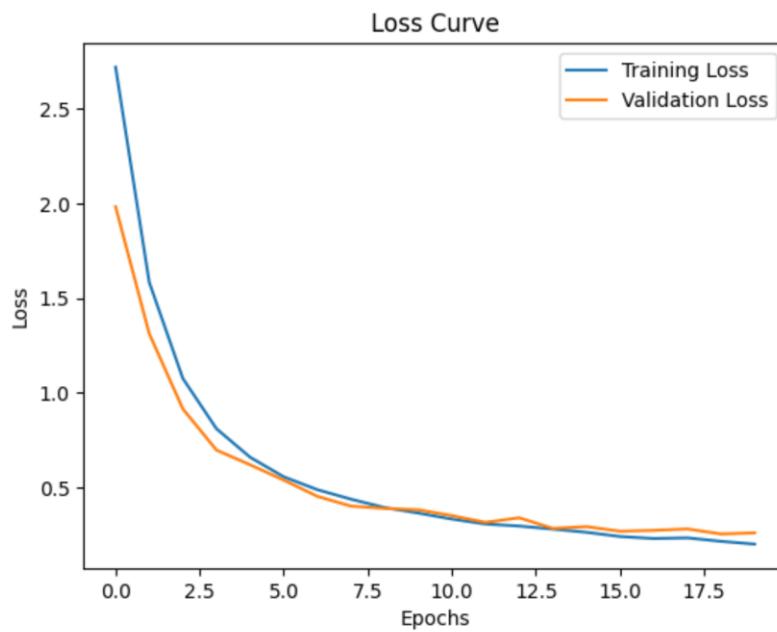
y_pred2 = model.predict(Valid_images)

y_pred2 = np.argmax(y_pred2, axis=1)
Valid_labels=np.argmax(Valid_labels, axis=1)
cm2 = confusion_matrix(Valid_labels, y_pred2)

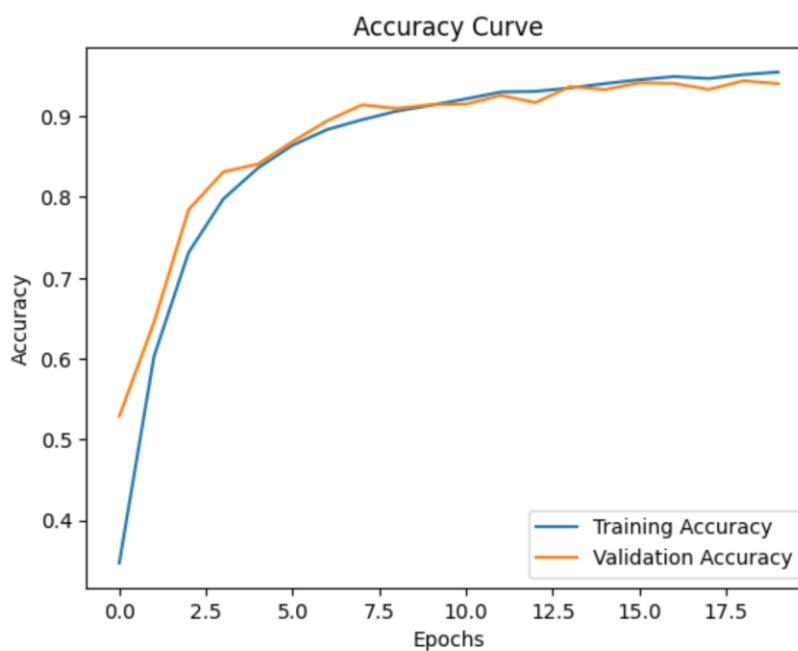
plt.figure(figsize=(14,10))
sns.heatmap(cm2, annot=True, fmt='d')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

```

مطابق کد بالا خطای دقت و ارزیابی **confusion Matrix** برای داده های تست و آورده و رسم می شود. این کد برای همه مدل های تربیت شده با شبکه پیچشی نیز مورد استفاده قرار می گیرد لذا از تکرار آن در مراحل بعد خود داری شده و فقط نتایج نشان داده شده اند.



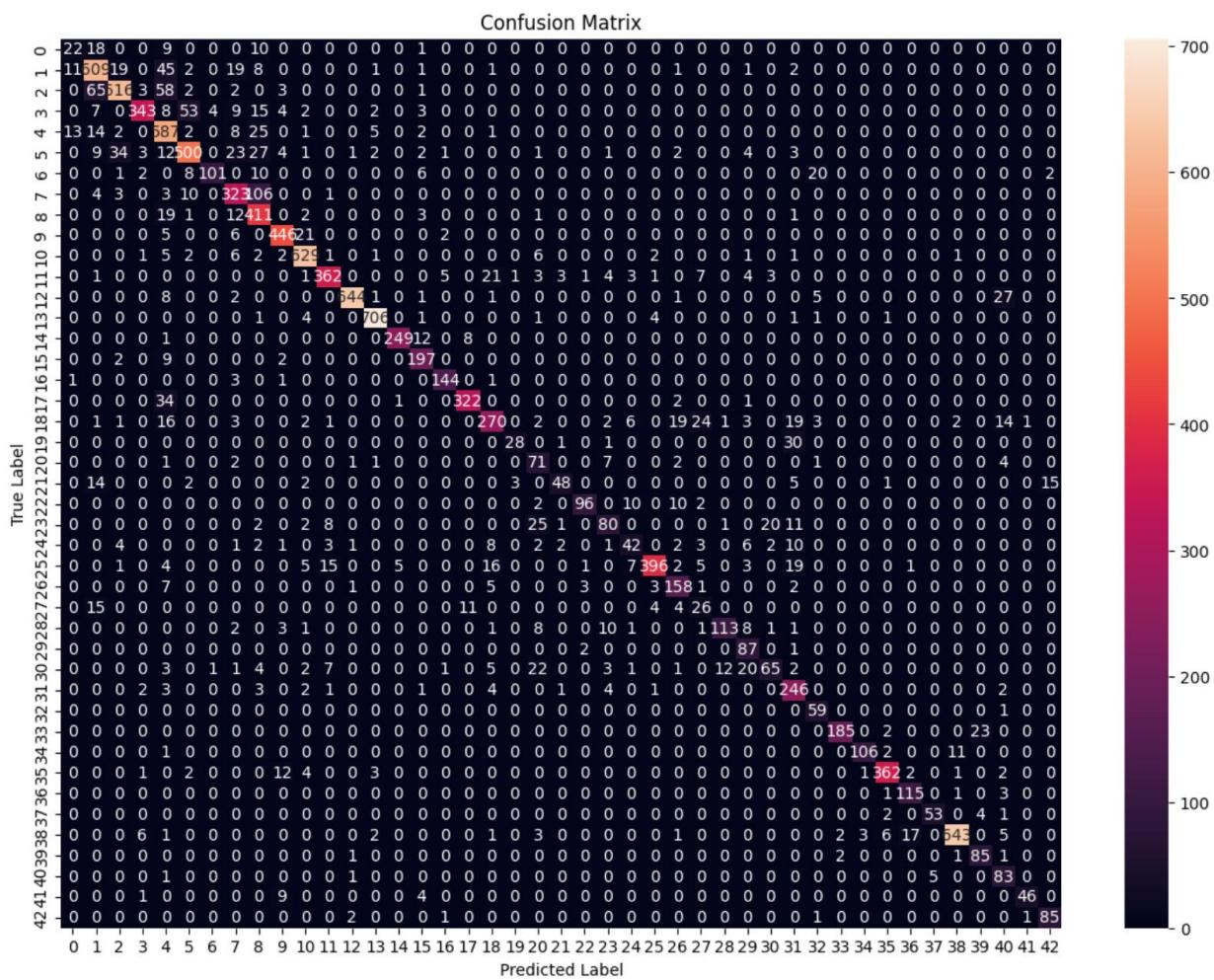
بر حسب تعداد ایپاک Loss function



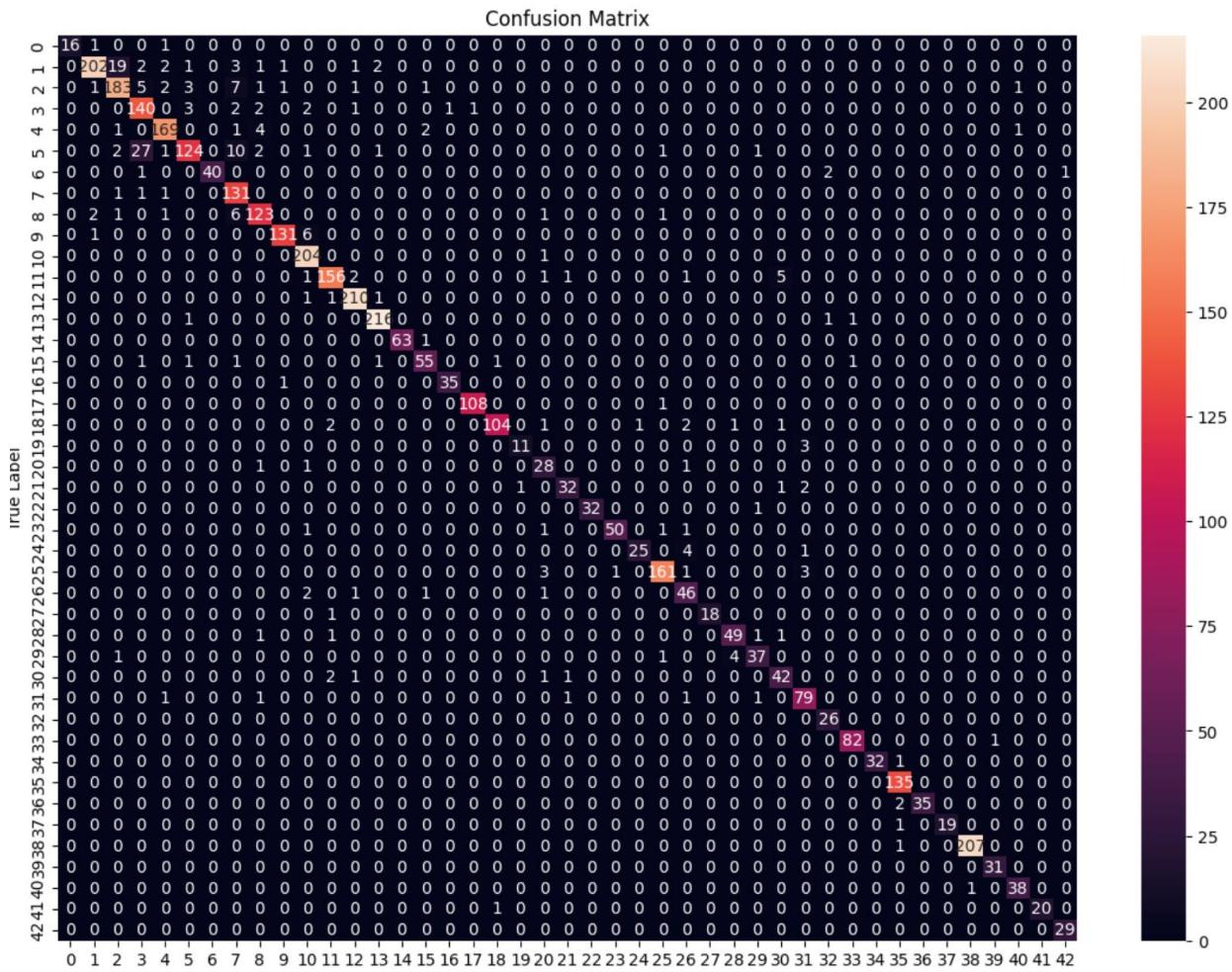
بر حسب تعداد ایپاک Accuracy

```
Test Loss: 0.7516814470291138
Test Accuracy: 0.8518606424331665
Validation Loss: 0.2587437629699707
Validation Accuracy: 0.9403213262557983
```

## دقت و خطای داده های تست و ارزیابی



## Confusion matrix برای داده های تست



## Confusion matrix برای داده های ارزیابی

# شبکه پیچشی

## آموزش شبکه

الف) شبکه پیچشی را در حالت‌های زیر آموزش دهید. در هر حالت می‌توانید تغییراتی را که باعث بهبود عملکرد شبکه شده است به حالت بعدی انتقال دهید (انجام این کار اجباری نیست).

- استفاده از pooling های مختلف در ساختار شبکه (دوحالت)
- استفاده یا عدم استفاده از dropout در آموزش شبکه (دوحالت)
- استفاده از بهینه سازهای Gradient Descent و Adam (دوحالت)
- استفاده از توابع فعال‌سازی tanh ، ReLU ، sigmoid (سه حالت)

```

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(30, 30, 3)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(43, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()

history = model.fit(Train_images, Train_labels, epochs=20, validation_data=(Valid_images, Valid_labels))

test_loss, test_acc = model.evaluate(Test_images, Test_labels)
print('Test accuracy:', test_acc)
print('Test loss:', test_loss)

model.save('model.h5')

netron.start('model.h5')

```

برای تعریف شبکه پیچشی :

- دو لایه پیچشی یکی با ۶۴ نورون و دیگری با ۳۲ نورون و یک لایه پنهان با ۶۴ نورون تعریف می کنیم.

- یک لایه flatten برای تبدیل تصاویر با بردار سه بعدی، به بردارهای یک بعدی تعریف میکنیم

- تعداد نورون لایه خروجی را برابر با تعداد خروجی ها در نظر میگیریم

- تابع فعال ساز لایه خروجی و بهینه ساز نیز متغیرهای سوال می باشند که در ادامه با انواع آن ها مدل را تربیت می کنیم.

- برای پلاس کردن مدل از تابع netron استفاده می کنیم

در ادامه و با توجه به خواسته های سوال در هر مرحله متغیرهای گفته شده را تغییر می دهیم ولی معماری شبکه در همه موارد مانند یکدیگر می باشد.

## شبکه پیچشی با Max pooling

### شبکه پیچشی Max pooling

```
# from tensorflow.keras.models import Sequential
# from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
# from tensorflow.keras.models import load_model
# import netron

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(30, 30, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(43, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()

history = model.fit(Train_images, Train_labels, epochs=20, validation_data=(Valid_images, Valid_labels))

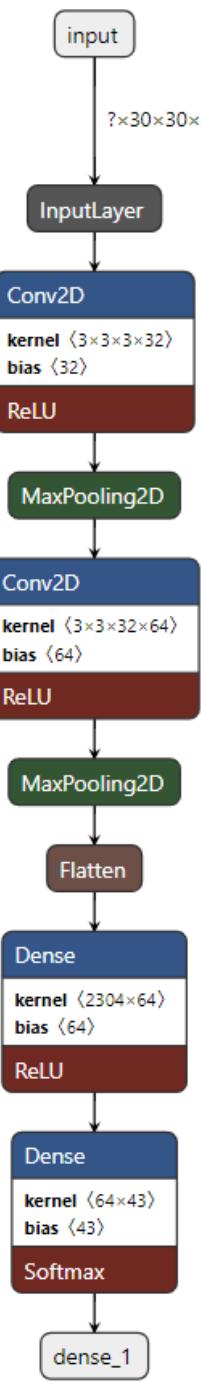
test_loss, test_acc = model.evaluate(Test_images, Test_labels)
print('Test accuracy:', test_acc)
print('Test loss:', test_loss)

model.save('model.h5')

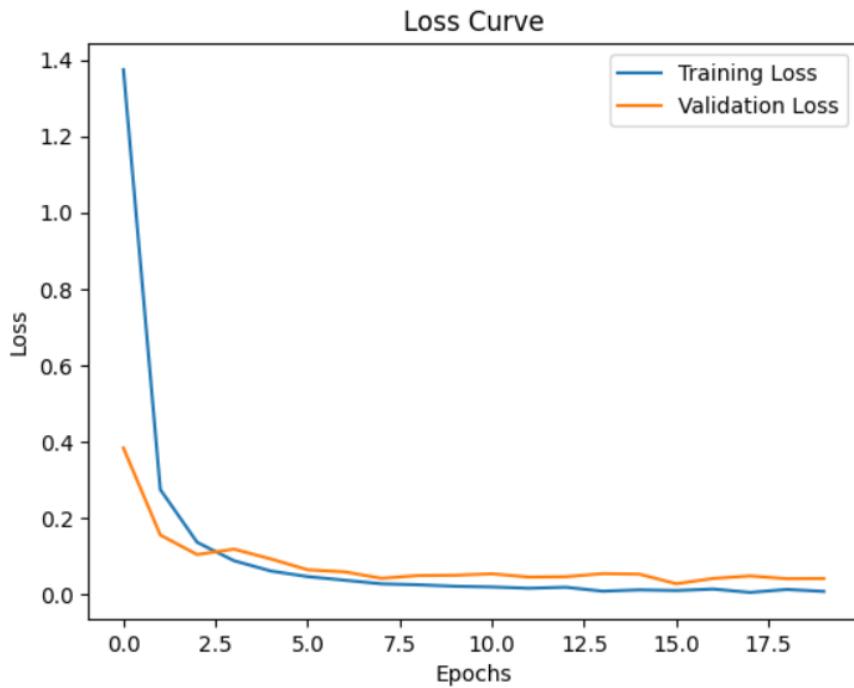
netron.start('model.h5')
```

Model: "sequential"

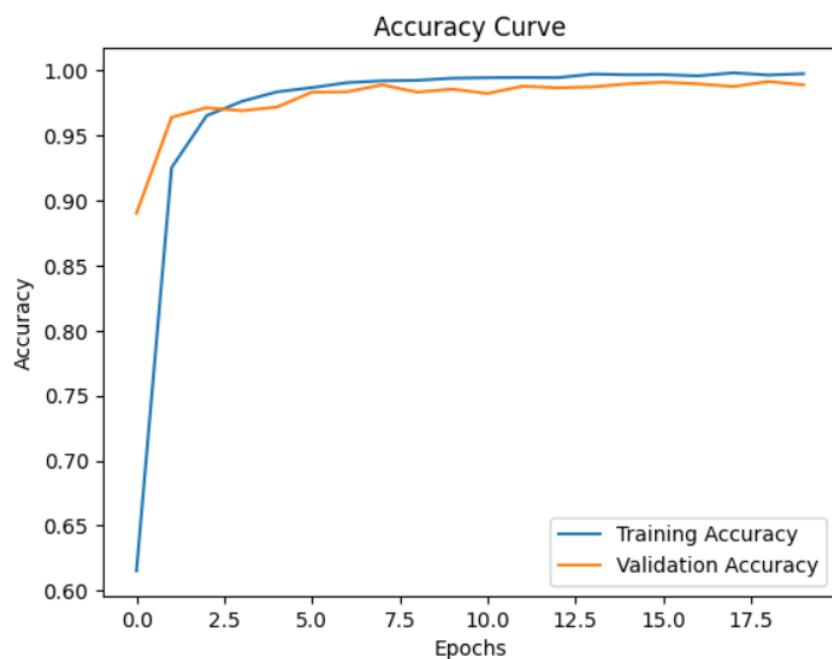
Layer (type)	Output Shape	Param #
<hr/>		
conv2d (Conv2D)	(None, 28, 28, 32)	896
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 12, 12, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 64)	147520
dense_1 (Dense)	(None, 43)	2795
<hr/>		
Total params: 169,707		
Trainable params: 169,707		
Non-trainable params: 0		



Max pooling برای شبکه پیچشی با Plot



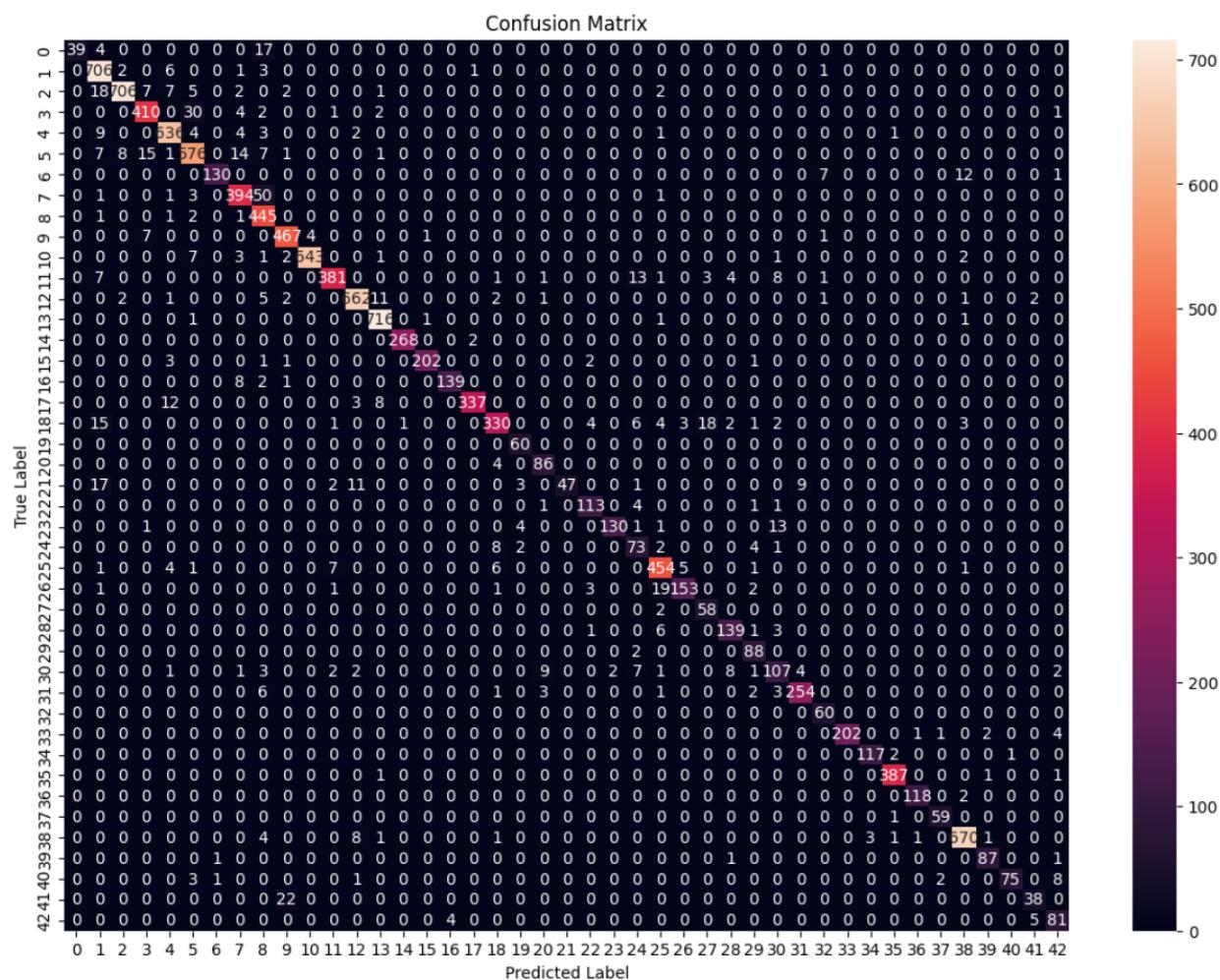
بر حسب تعداد ایپاک Loss function



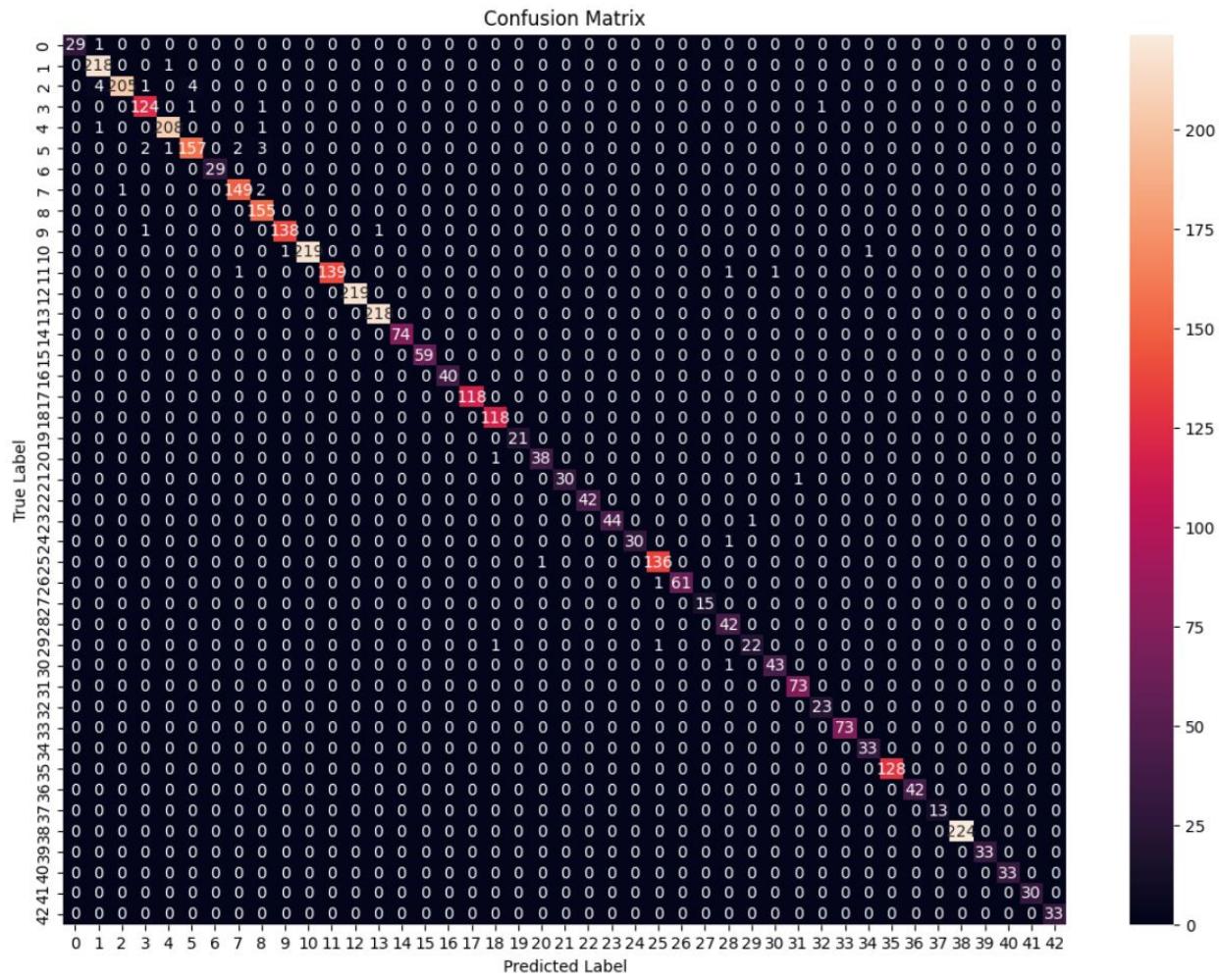
بر حسب تعداد ایپاک Accuracy

```
Test Loss: 0.48618727922439575  
Test Accuracy: 0.9376880526542664  
Validation Loss: 0.04198436439037323  
Validation Accuracy: 0.9890334010124207
```

## دقت و خطأ برای داده های تست و ارزیابی



## Confusion Matrix برای داده های تست



## Confusion Matrix برای داده های ارزیابی

## شبکه پیچشی با Average pooling

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, AveragePooling2D, Flatten, Dense
from tensorflow.keras.models import load_model
import netron

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(30, 30, 3)))
model.add(AveragePooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(AveragePooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(43, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()

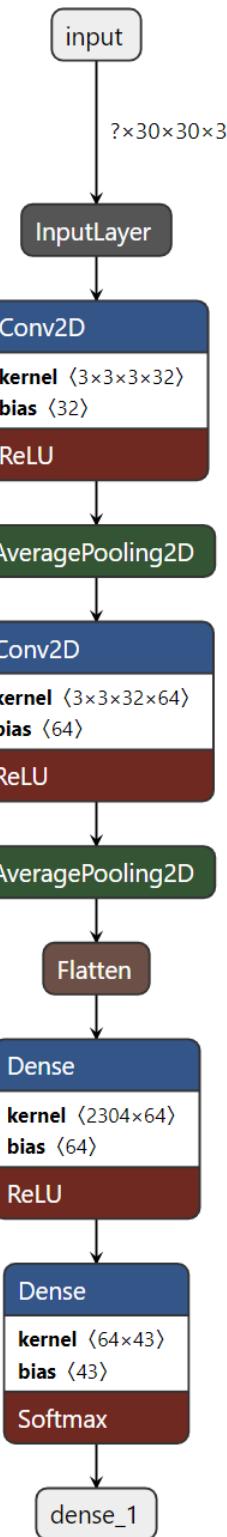
history = model.fit(Train_images, Train_labels, epochs=20, validation_data=(Valid_images, Valid_labels))

test_loss, test_acc = model.evaluate(Test_images, Test_labels)
print('Test accuracy:', test_acc)
print('Test loss:', test_loss)

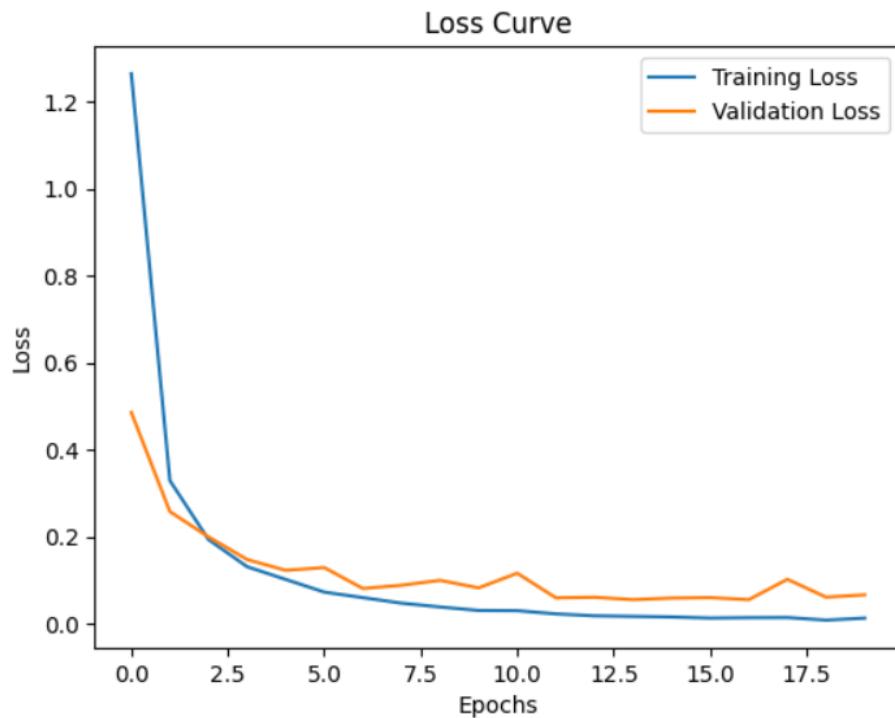
model.save('model.h5')

netron.start('model.h5')
```

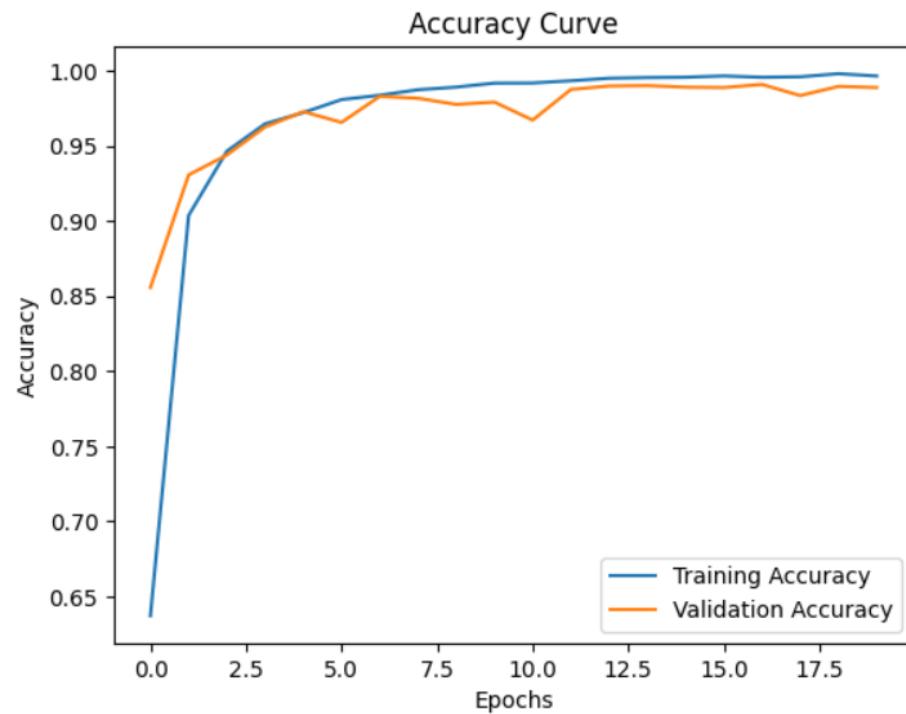
```
Model: "sequential"
=====
Layer (type)          Output Shape         Param #
=====
conv2d (Conv2D)        (None, 28, 28, 32)      896
average_pooling2d (AverageP (None, 14, 14, 32)      0
ooling2D)
conv2d_1 (Conv2D)       (None, 12, 12, 64)     18496
average_pooling2d_1 (Averag (None, 6, 6, 64)      0
ePooling2D)
flatten (Flatten)       (None, 2304)           0
dense (Dense)          (None, 64)             147520
dense_1 (Dense)         (None, 43)            2795
=====
Total params: 169,707
Trainable params: 169,707
Non-trainable params: 0
```



Average pooling Plot برای شبکه عصبی



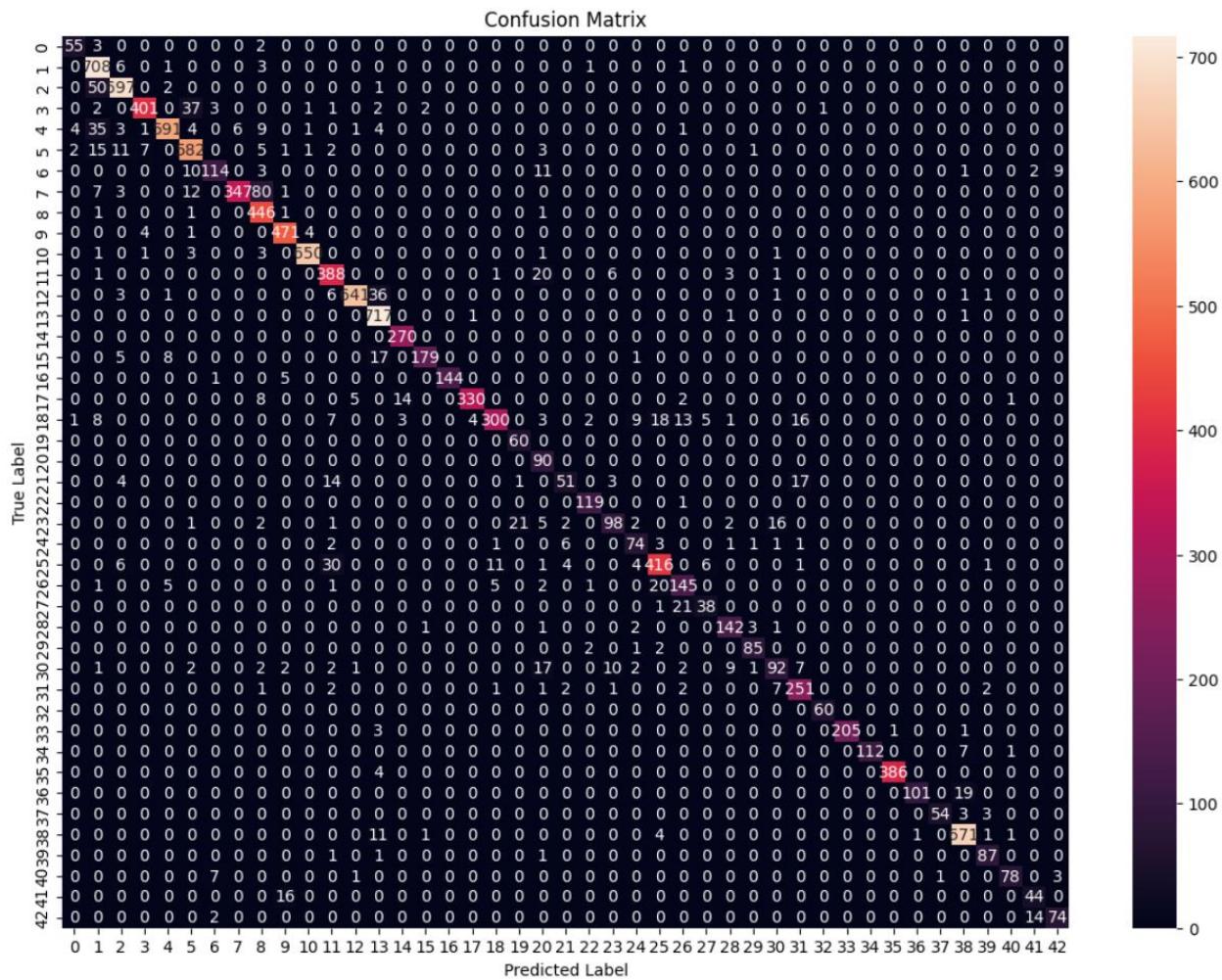
بر حسب تعداد ایپاک **Loss function**



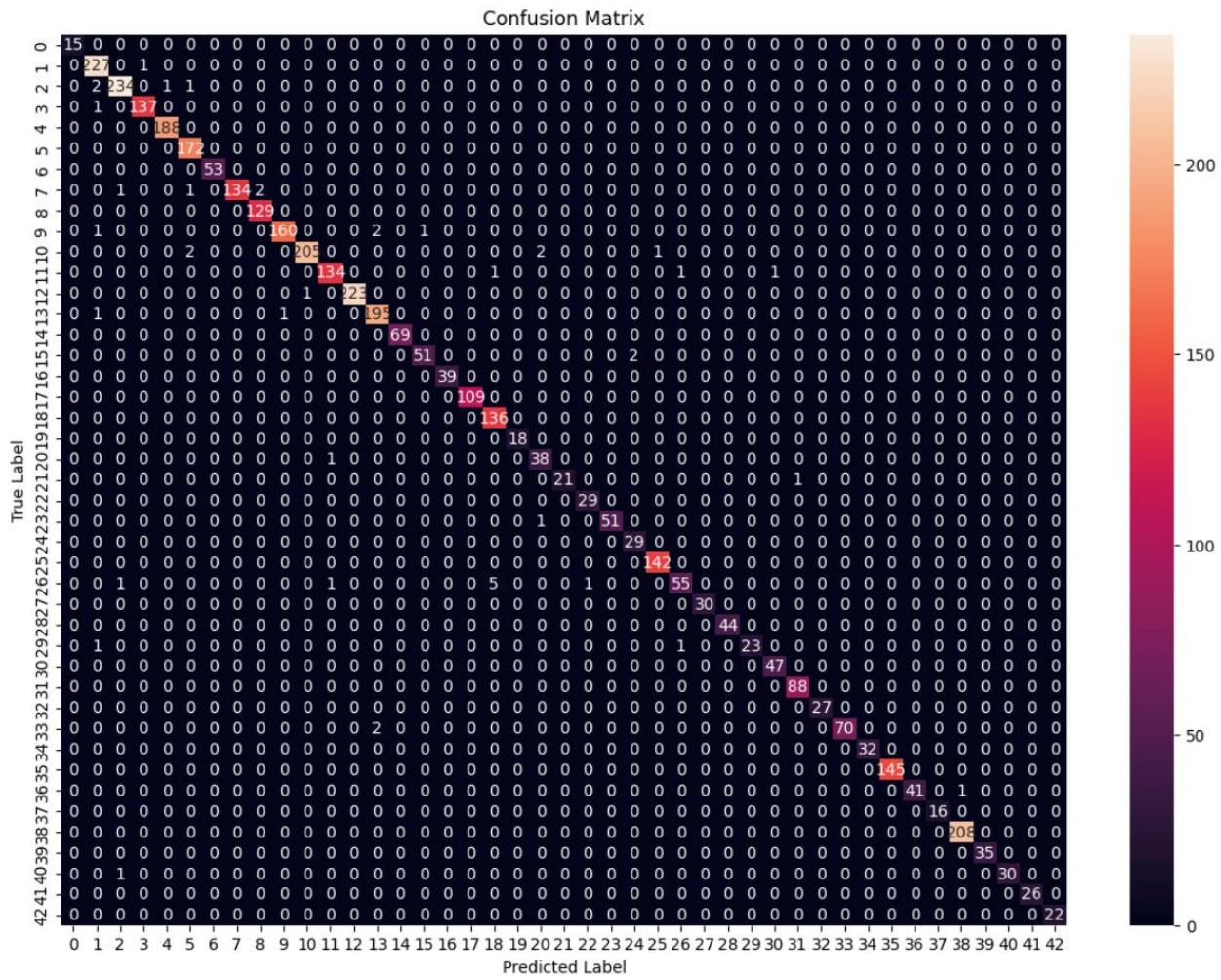
بر حسب تعداد ایپاک **Accuracy**

```
Test Loss: 0.5596073865890503  
Test Accuracy: 0.9155977964401245  
Validation Loss: 0.06614147871732712  
Validation Accuracy: 0.9887783527374268
```

## دقت و خطأ برای داده های تست و ارزیابی



## Confusion Matrix برای داده های تست



## Confusion Matrix برای داده های ارزیابی

## شبکه پیچشی با : Dropout

از آنجایی که تربیت شبکه با Max pooling موجب بهبود شبکه شد مدل بدون dropout ما همان مدل است و در اینجا مدل با استفاده از drop out را بررسی می کنیم.

### شبکه پیچشی با dropout

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import load_model
import netron

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(30, 30, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(43, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()

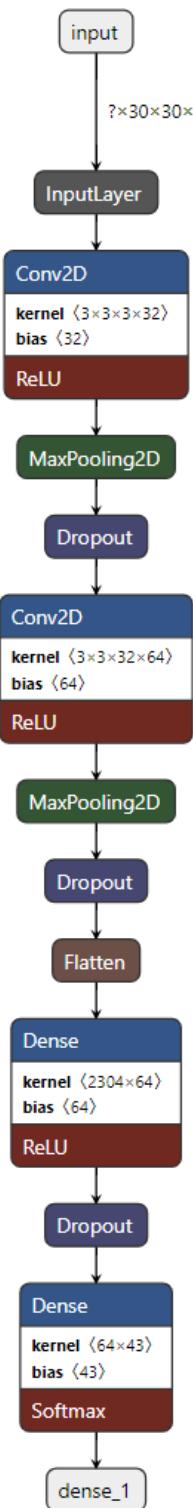
history = model.fit(Train_images, Train_labels, epochs=20, validation_data=(Valid_images, Valid_labels))

test_loss, test_acc = model.evaluate(Test_images, Test_labels)
print('Test accuracy:', test_acc)
print('Test loss:', test_loss)

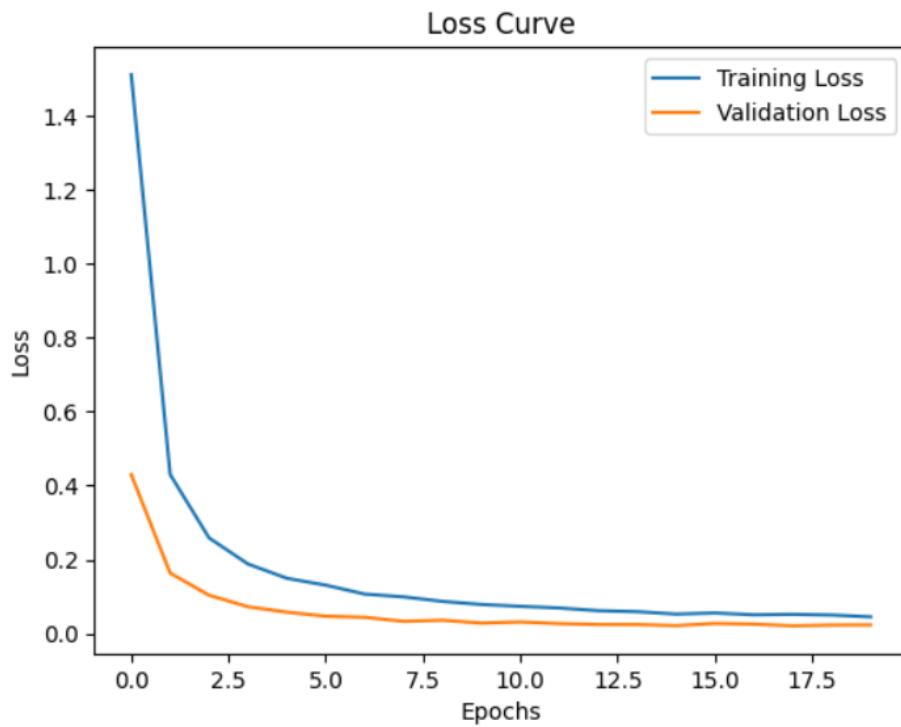
model.save('model.h5')

netron.start('model.h5')
```

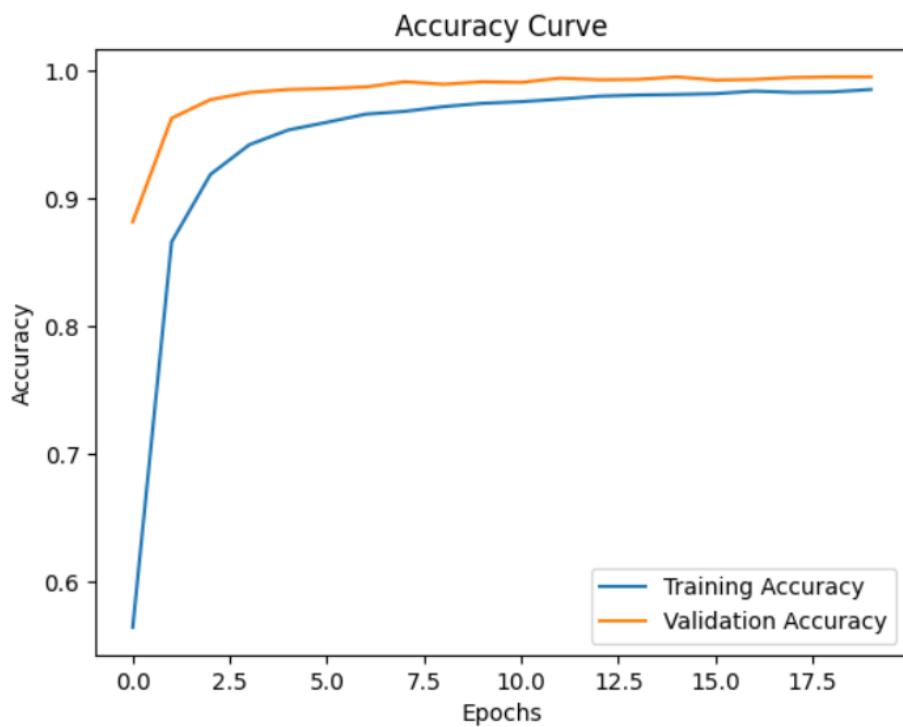
Model: "sequential"		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	896
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
dropout (Dropout)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 12, 12, 64)	18496
max_pooling2d_1 (MaxPooling 2D)	(None, 6, 6, 64)	0
dropout_1 (Dropout)	(None, 6, 6, 64)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 64)	147520
dropout_2 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 43)	2795
<hr/>		
Total params: 169,707		
Trainable params: 169,707		
Non-trainable params: 0		



Dropout شبکه پیچشی با Plot



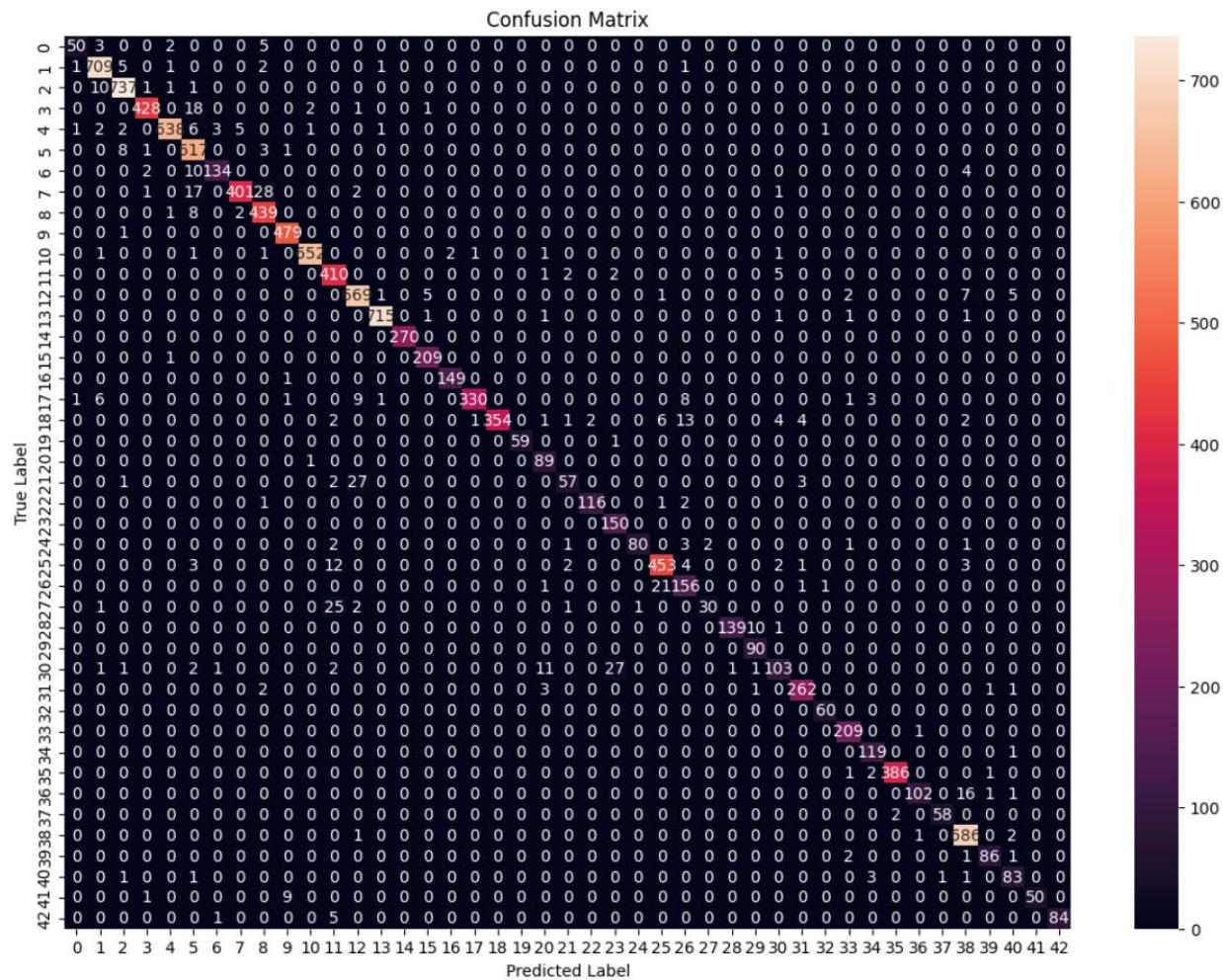
بر حسب تعداد ایپاک Loss function



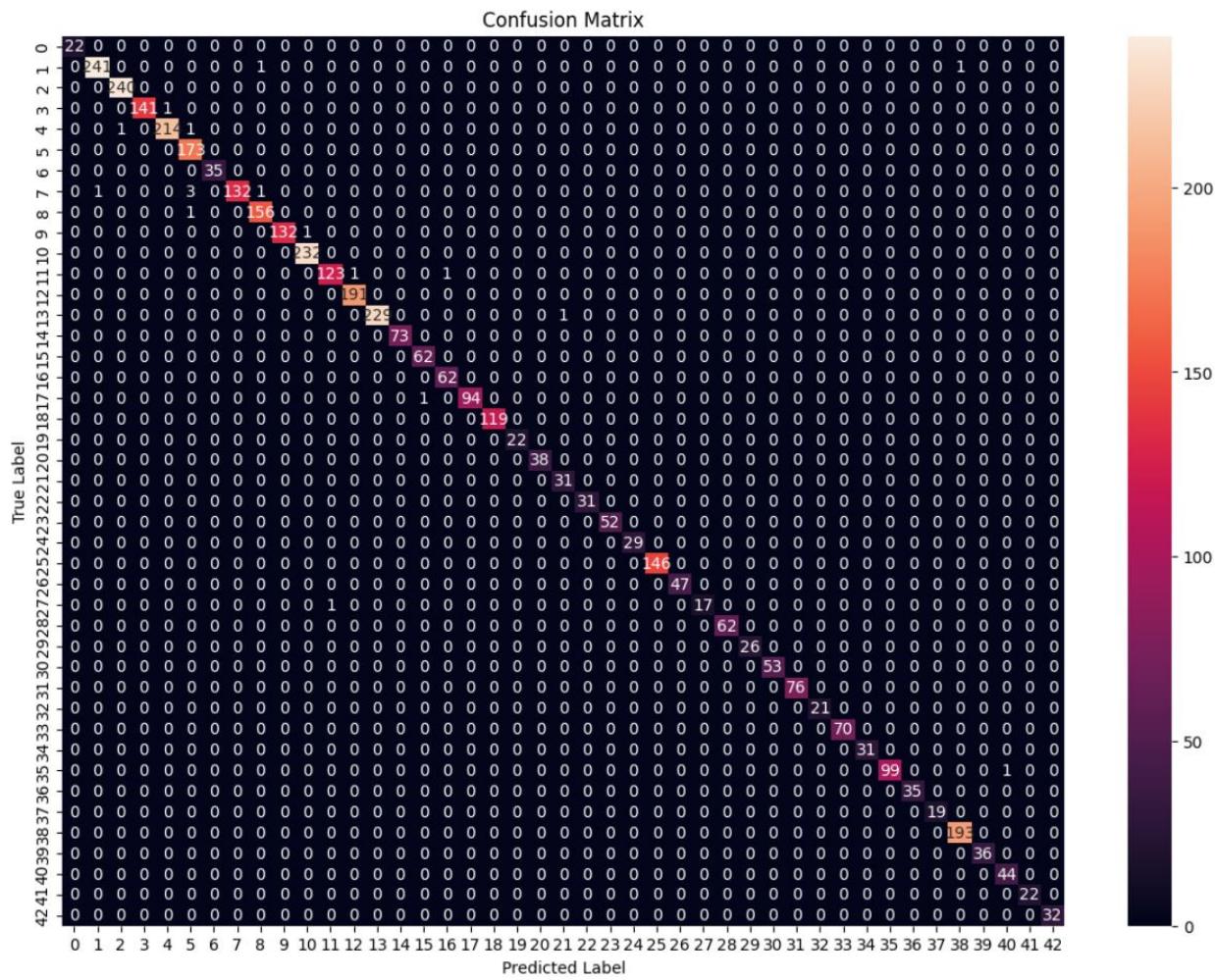
بر حسب تعداد ایپاک Accuracy

Test Loss: 0.2159057855606079  
 Test Accuracy: 0.9577988982200623  
 Validation Loss: 0.02376691810786724  
 Validation Accuracy: 0.9954093098640442

## دقت و خطای برای داده های تست و ارزیابی



Confusion Matrix برای داده های تست



## Confusion Matrix برای داده های ارزیابی

## شبکه پیچشی با بهینه ساز SGD :

بایوچه به اینکه dropout نمودار دقت و خطای smooth یا یکنواخت کرد آن را نگه می داریم و برای مدل همان مدل dropout را در نظر می گیریم.

### شبکه پیچشی SGD

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import load_model
import netron

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(30, 30, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(43, activation='softmax'))

model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()

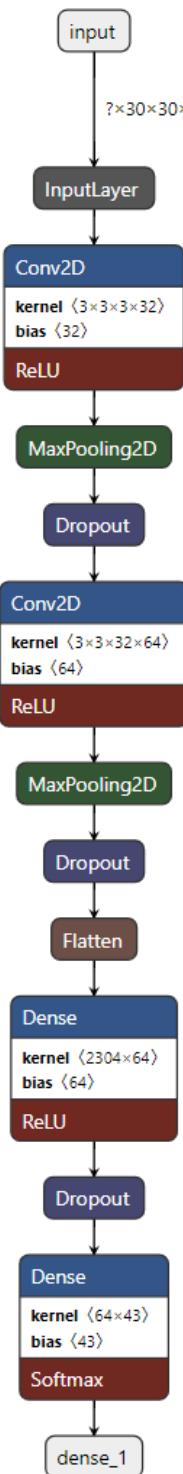
history = model.fit(Train_images, Train_labels, epochs=20, validation_data=(Valid_images, Valid_labels))

test_loss, test_acc = model.evaluate(Test_images, Test_labels)
print('Test accuracy:', test_acc)
print('Test loss:', test_loss)

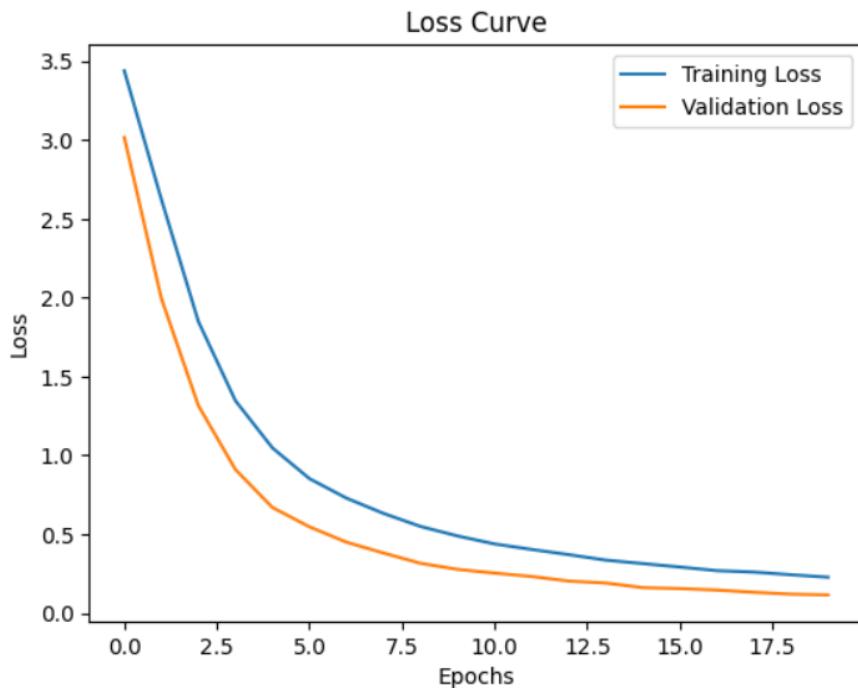
model.save('model.h5')

netron.start('model.h5')

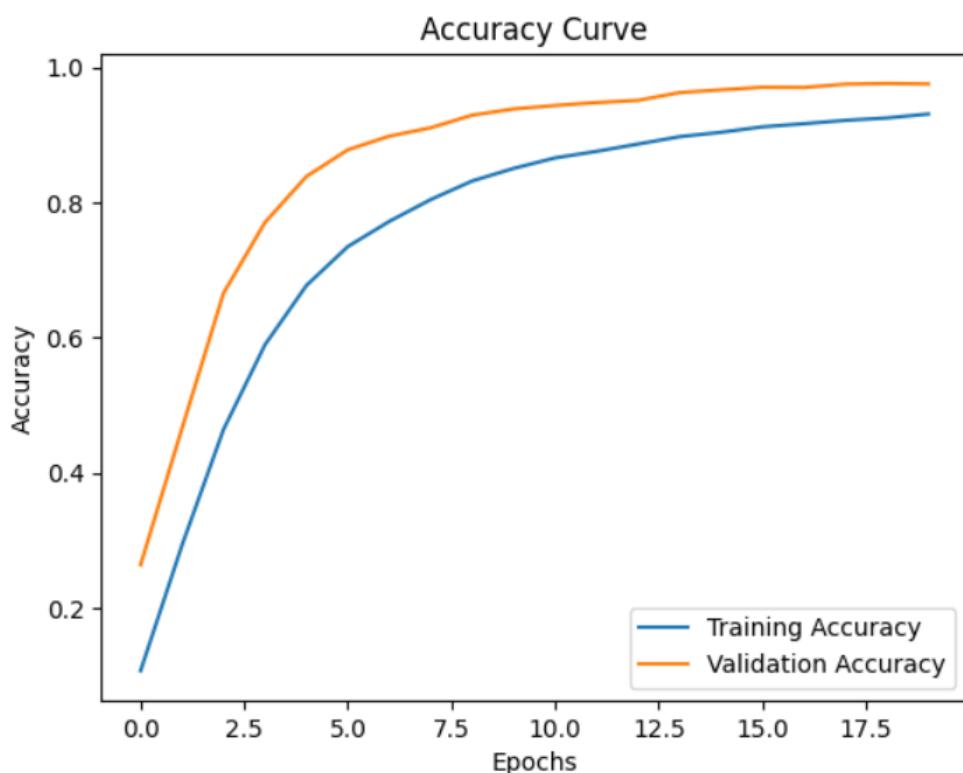
Model: "sequential"
=====
Layer (type)          Output Shape         Param #
=====
conv2d (Conv2D)        (None, 28, 28, 32)      896
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)    0
dropout (Dropout)      (None, 14, 14, 32)      0
conv2d_1 (Conv2D)       (None, 12, 12, 64)     18496
max_pooling2d_1 (MaxPooling2D) (None, 6, 6, 64)    0
dropout_1 (Dropout)     (None, 6, 6, 64)      0
flatten (Flatten)       (None, 2304)           0
dense (Dense)          (None, 64)            147520
dropout_2 (Dropout)     (None, 64)            0
dense_1 (Dense)         (None, 43)            2795
=====
Total params: 169,707
Trainable params: 169,707
Non-trainable params: 0
```



برای شبکه پیچشی با بهینه ساز SGD Plot



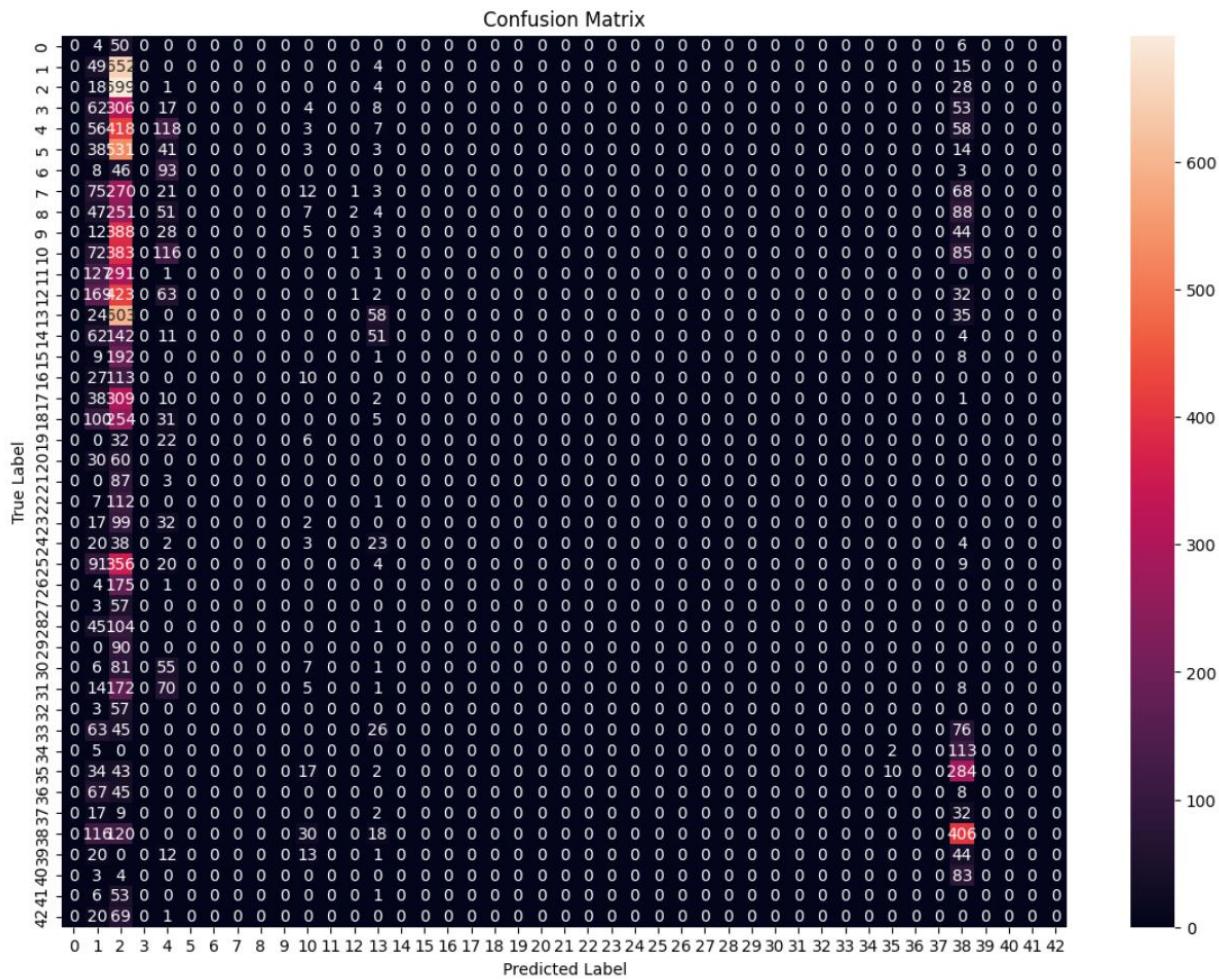
بر حسب تعداد ایپاک **Loss function**



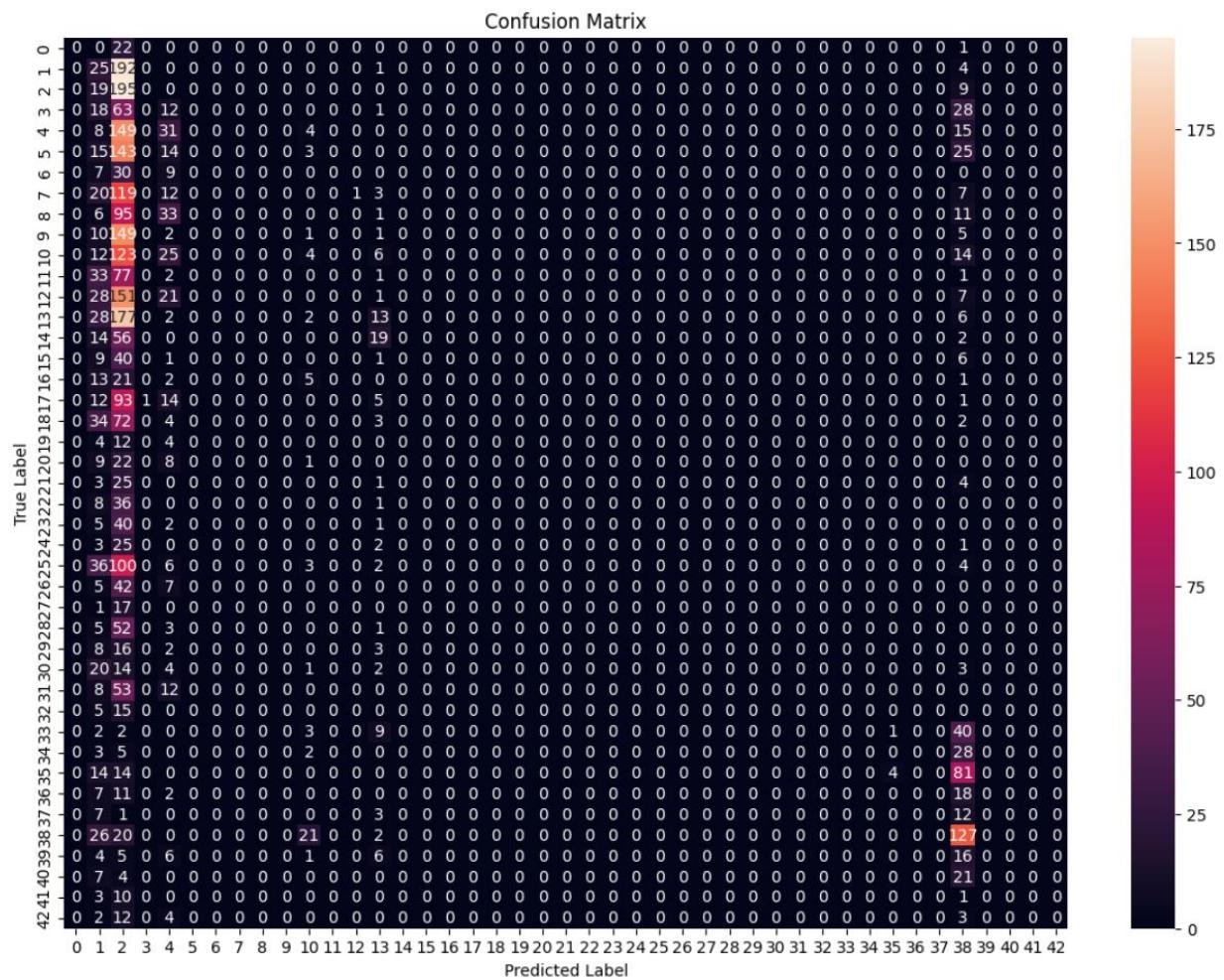
بر حسب تعداد ایپاک **Accuracy**

Test Loss: 0.36869513988494873  
 Test Accuracy: 0.9092636704444885  
 Validation Loss: 0.11457298696041107  
 Validation Accuracy: 0.9755164384841919

## دقت و خطای برای داده های تست و ارزیابی



Confusion matrix برای داده های تست



## Confusion matrix برای داده های ارزیابی

## شبکه پیچشی با تابع فعال سازی : Relu

تا اینجا همه‌ی شبکه‌های پیچشی تربیت شده با تابع فعال ساز Relu برای تمام حالت‌ها تربیت شدند لذا برای شبکه پیچشی Relu داده‌های همان‌ها را استفاده می‌کنیم.

همچنین تا اینجا بهترین مدل مربوط به dropout و بهینه‌ساز Adam با Max pooling بوده است لذا با این توابع ادامه می‌دهیم.

## شبکه پیچشی با تابع فعال سازی sigmoid

### شبکه پیچشی با تابع فعال سازی sigmoid

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import load_model
import netron

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='sigmoid', input_shape=(30, 30, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='sigmoid'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(64, activation='sigmoid'))
model.add(Dropout(0.2))
model.add(Dense(43, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()

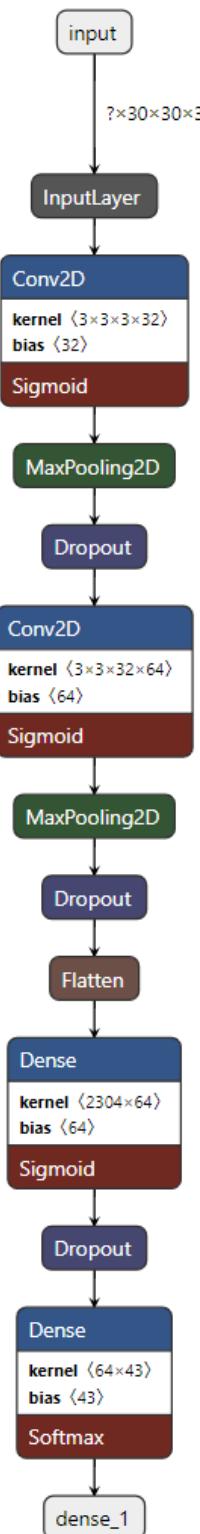
history = model.fit(Train_images, Train_labels, epochs=20, validation_data=(Valid_images, Valid_labels))

test_loss, test_acc = model.evaluate(Test_images, Test_labels)
print('Test accuracy:', test_acc)
print('Test loss:', test_loss)

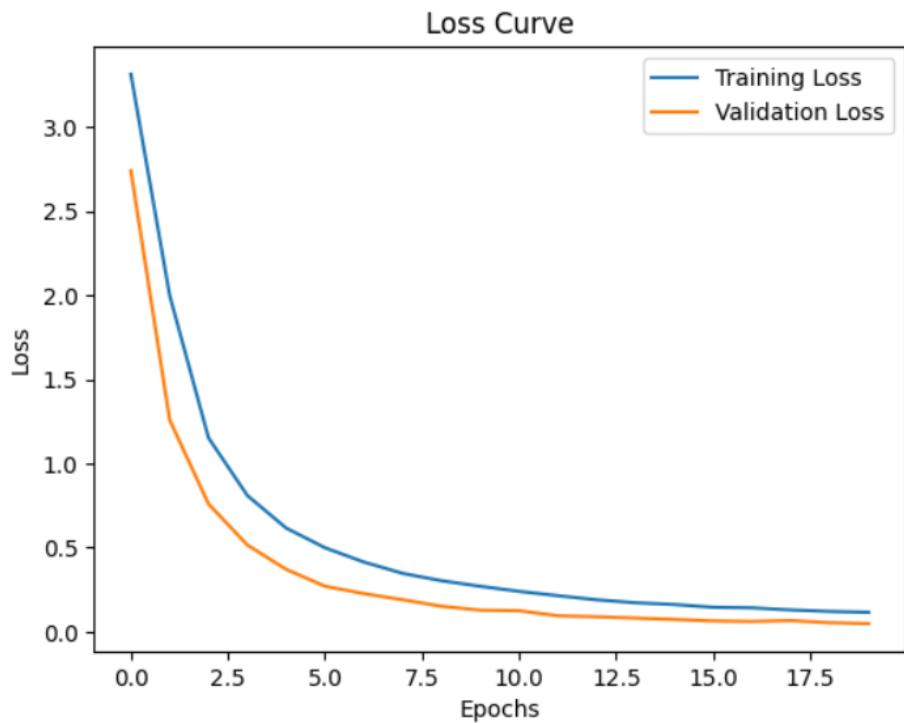
model.save('model.h5')

netron.start('model.h5')
```

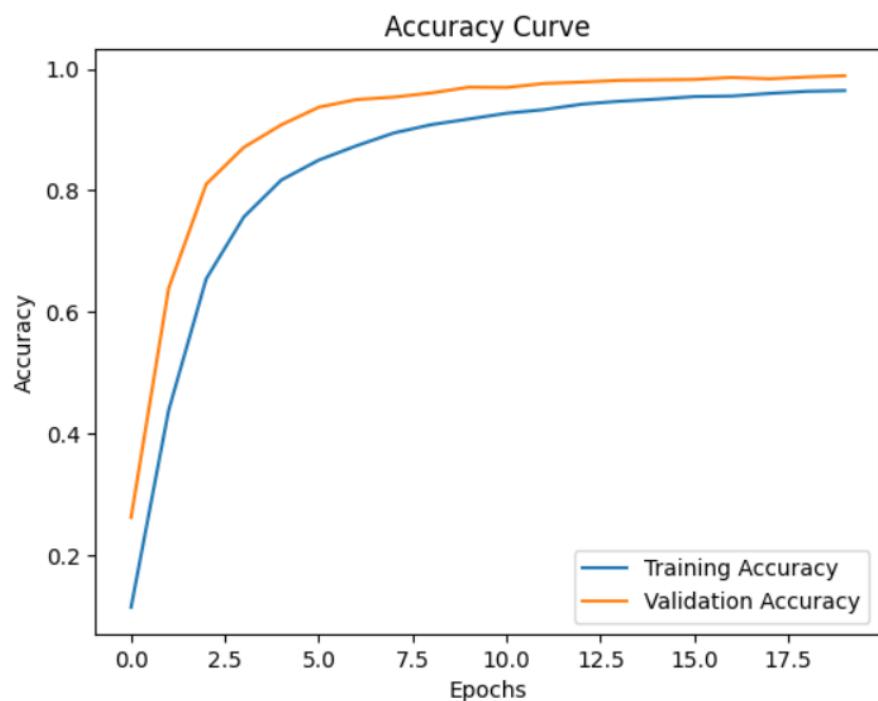
```
Model: "sequential"
=====
Layer (type)          Output Shape         Param #
=====
conv2d (Conv2D)        (None, 28, 28, 32)      896
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)    0
)
dropout (Dropout)      (None, 14, 14, 32)      0
conv2d_1 (Conv2D)       (None, 12, 12, 64)     18496
max_pooling2d_1 (MaxPooling2D) (None, 6, 6, 64)    0
)
dropout_1 (Dropout)     (None, 6, 6, 64)      0
flatten (Flatten)       (None, 2304)           0
dense (Dense)          (None, 64)            147520
dropout_2 (Dropout)     (None, 64)            0
dense_1 (Dense)         (None, 43)            2795
=====
Total params: 169,707
Trainable params: 169,707
Non-trainable params: 0
```



sigmoid برای شبکه پیچشی با تابع فعال ساز Plot



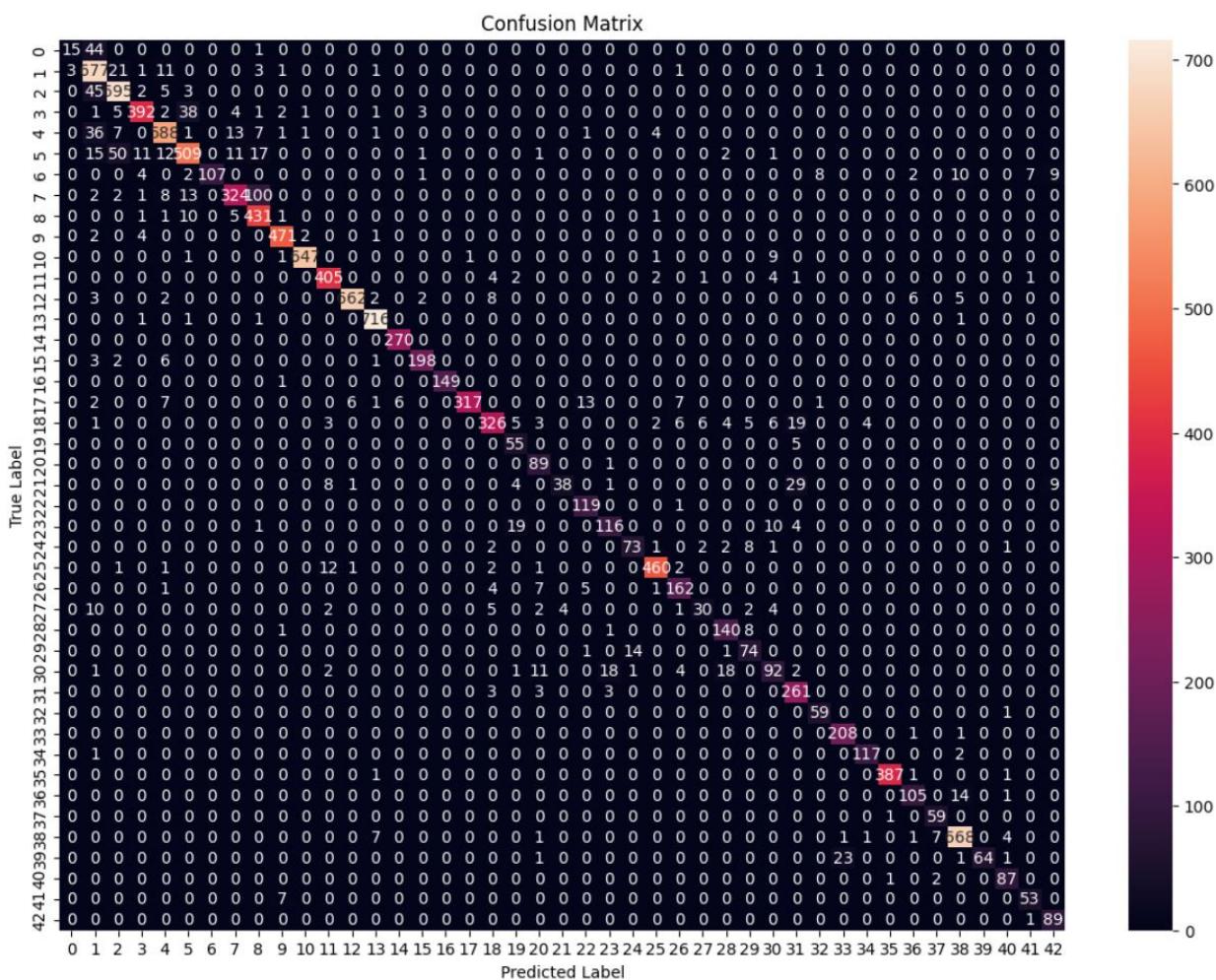
بر حسب تعداد ایپاک Loss function



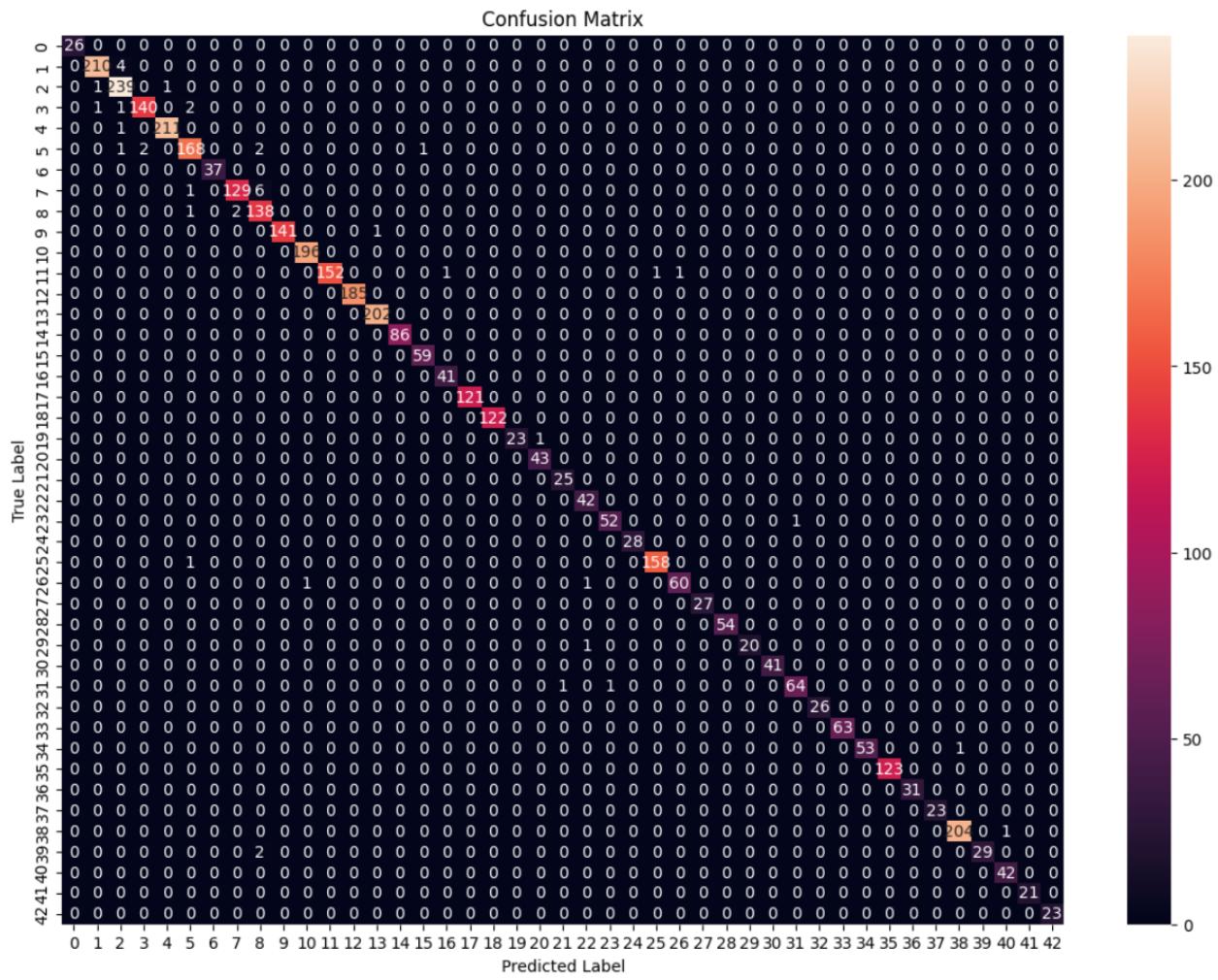
بر حسب تعداد ایپاک Accuracy

```
Test Loss: 0.30611008405685425  
Test Accuracy: 0.9108471870422363  
Validation Loss: 0.048046376556158066  
Validation Accuracy: 0.9890334010124207
```

## دقت و خطای داده های تست و ارزیابی



## برای داده های تست Confusion Matrix



## برای داده های تست Confusion Matrix

## شبکه پیچشی با تابع فعال سازی :tanh

### شبکه پیچشی با تابع فعال سازی tanh ¶

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import load_model
import netron

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='tanh', input_shape=(30, 30, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='tanh'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(64, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(43, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()

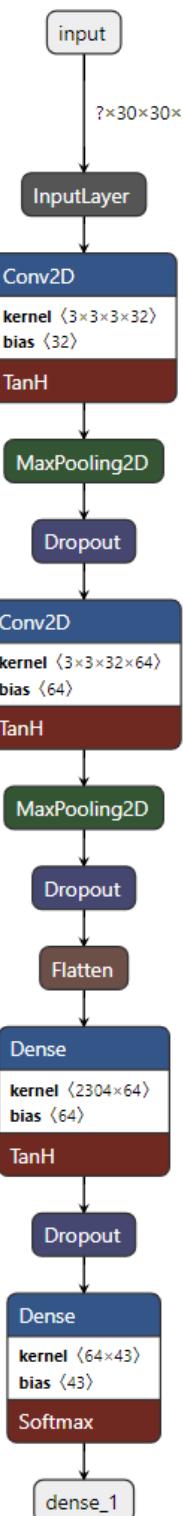
history = model.fit(Train_images, Train_labels, epochs=20, validation_data=(Valid_images, Valid_labels))

test_loss, test_acc = model.evaluate(Test_images, Test_labels)
print('Test accuracy:', test_acc)
print('Test loss:', test_loss)

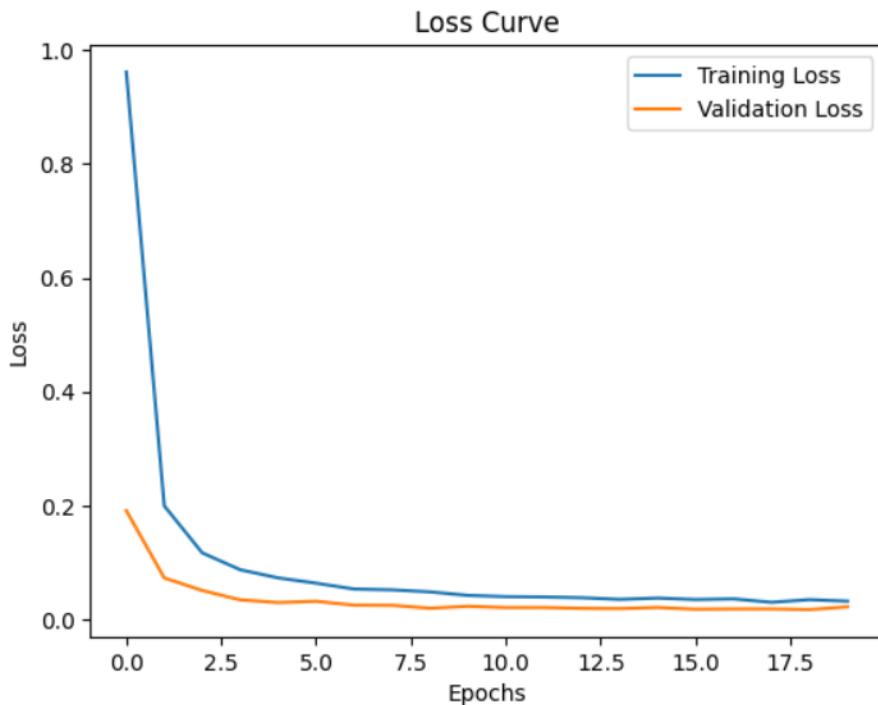
model.save('model.h5')

netron.start('model.h5')
```

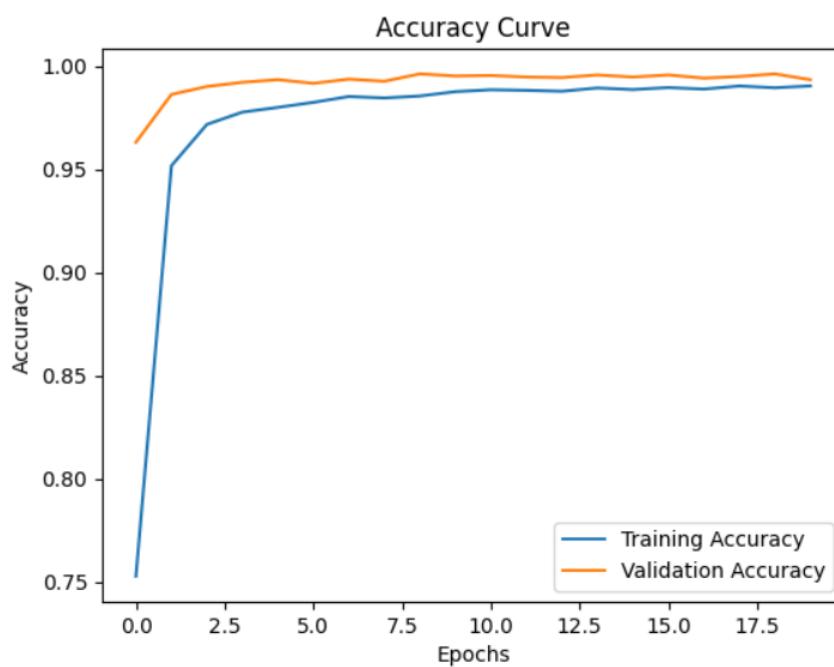
Model: "sequential"		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	896
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
dropout (Dropout)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 12, 12, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
dropout_1 (Dropout)	(None, 6, 6, 64)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 64)	147520
dropout_2 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 43)	2795
=====		
Total params: 169,707		
Trainable params: 169,707		
Non-trainable params: 0		



برای شبکه پیچشی با تابع فعال ساز **tanh** Plot



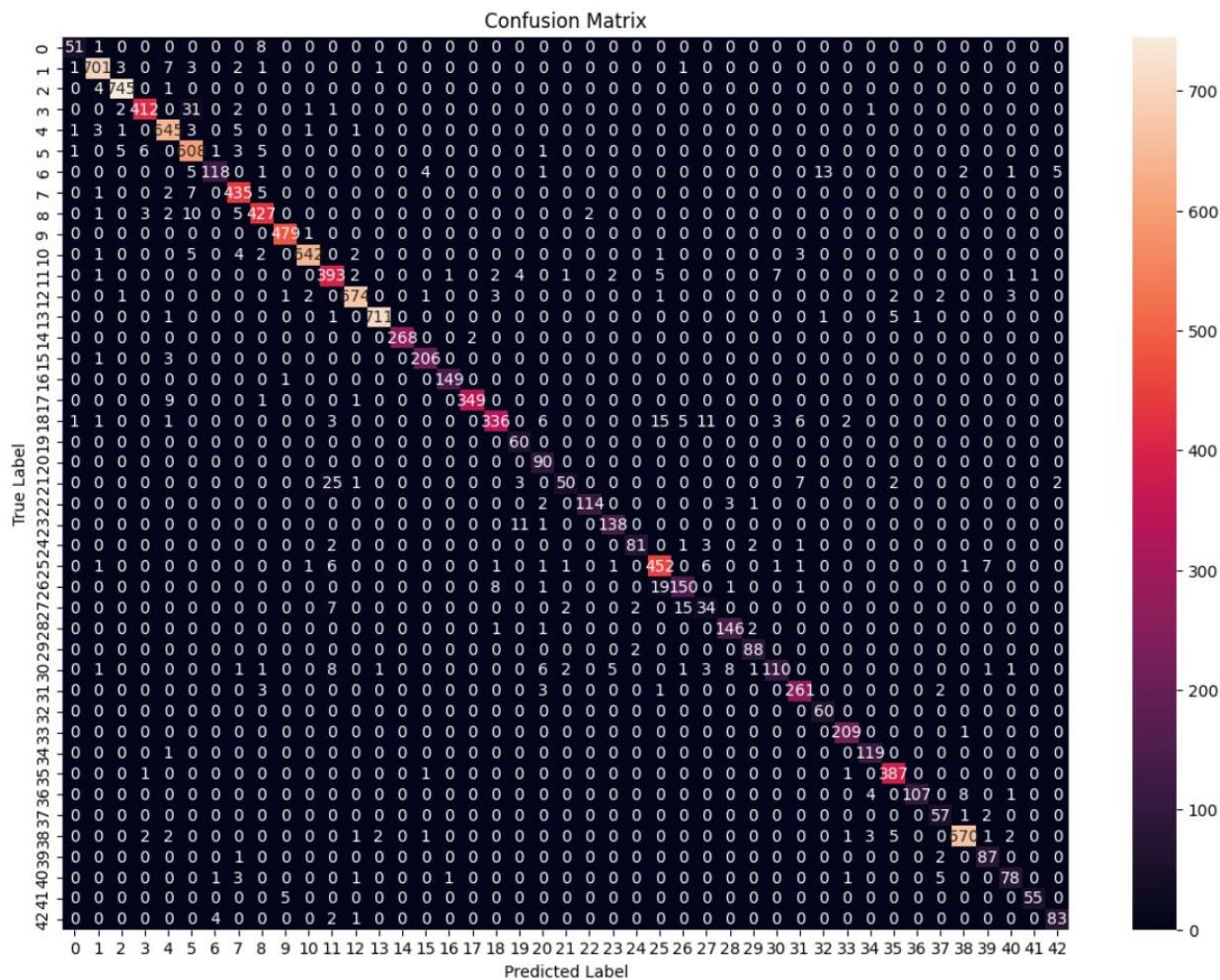
بر حسب تعداد ایپاک Loss function



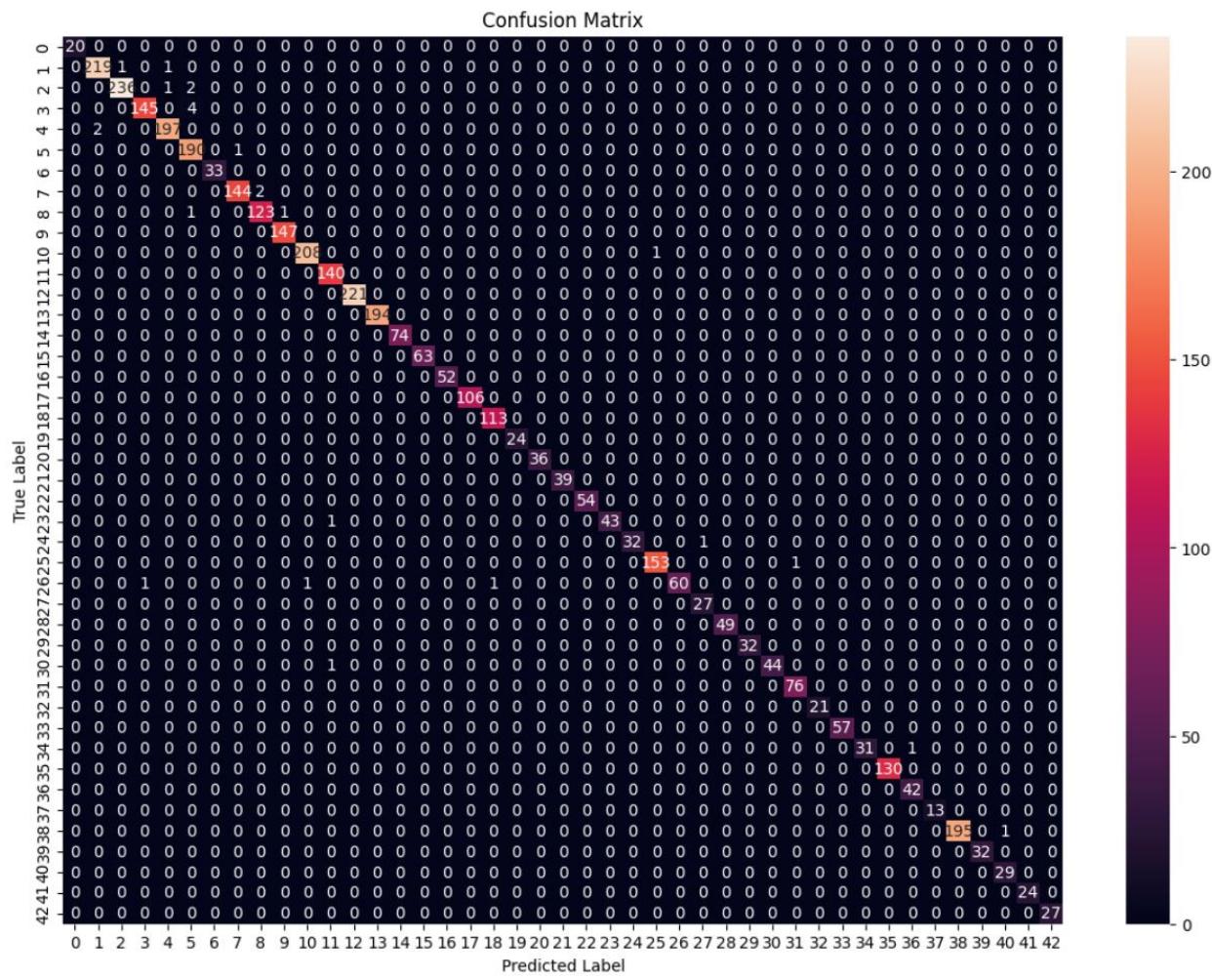
بر حسب تعداد ایپاک Accuracy

```
Test Loss: 0.18630479276180267  
Test Accuracy: 0.9528899192810059  
Validation Loss: 0.02172083966434002  
Validation Accuracy: 0.9933690428733826
```

## دقت و خطای داده های تست و ارزیابی



## Confusion Matrix برای داده های تست



## Confusion Matrix برای داده های ارزیابی

## شبکه پیچشی با روش : Data Augmentation

ب) ابتدا در مورد روش Data Augmentation و اهمیت آن در یادگیری عمیق توضیح دهید. سپس ساختاری را که در بخش قبل به بیشترین دقت دست یافته اینبار به کمک Data Augmentation آموزش دهید (حداقل از سه نوع نگاشت شناختی تصویری استفاده کنید). تو پیش دهید که با توجه به ماهیت مسئله، چه نگاشتهايی مجاز هستند (برخی از نگاشت ها باعث می شوند عکس یک تابلو در دسته نادرست طبقه بندی شود). برای پیاده سازی Data augmentation می توانید از تابع `ImageDataGenerator` در کتابخانه `keras.preprocessing.image` استفاده کنید. برای پیاده سازی این بخش می توانید از این [لينك](#) کمک بگیرید.

تا اینجا بهترین مدل شامل موارد زیر می باشد:

**Max pooling \_**

**Dropout \_**

**Adam \_ بهینه ساز**

**Relu \_ تابع فعال ساز**

لذا روش Data Augmentation را روی مدل با این مشخصات پیاده میکنیم.

## شبکه پیچشی با روش Data Augmentation

```
▶ from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import load_model
import netron
from tensorflow.keras.preprocessing.image import ImageDataGenerator

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(30, 30, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(43, activation='softmax'))

#rotation
datagen = ImageDataGenerator(rotation_range=30, fill_mode='nearest')
aug_iter = datagen.flow(Train_images, batch_size=1)

fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(15,15))

for i in range(3):
    image = next(aug_iter)[0].astype('uint8')
    ax[i].imshow(image)
    ax[i].axis('off')

#shifting
datagen2 = ImageDataGenerator(width_shift_range=0.2, height_shift_range=0.2)
aug_iter = datagen2.flow(Train_images, batch_size=1)

fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(15,15))

for i in range(3):
    image = next(aug_iter)[0].astype('uint8')
    ax[i].imshow(image)
    ax[i].axis('off')

#brightness
datagen3 = ImageDataGenerator(brightness_range=[0.4,1.5])
aug_iter = datagen3.flow(Train_images, batch_size=1)

fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(15,15))

for i in range(3):
    image = next(aug_iter)[0].astype('uint8')
    ax[i].imshow(image)
    ax[i].axis('off')
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()

history = model.fit(Train_images, Train_labels, epochs=20, validation_data=(Valid_images, Valid_labels))

test_loss, test_acc = model.evaluate(Test_images, Test_labels)
print('Test accuracy:', test_acc)
print('Test loss:', test_loss)

model.save('model.h5')

netron.start('model.h5')
```

```

Model: "sequential_3"

Layer (type)          Output Shape       Param #
=====
conv2d_6 (Conv2D)      (None, 28, 28, 32)    896
max_pooling2d_6 (MaxPooling 2D)        (None, 14, 14, 32)    0
dropout_9 (Dropout)     (None, 14, 14, 32)    0
conv2d_7 (Conv2D)       (None, 12, 12, 64)   18496
max_pooling2d_7 (MaxPooling 2D)        (None, 6, 6, 64)    0
dropout_10 (Dropout)    (None, 6, 6, 64)    0
flatten_3 (Flatten)    (None, 2304)         0
dense_6 (Dense)        (None, 64)           147520
dropout_11 (Dropout)    (None, 64)           0
dense_7 (Dense)        (None, 43)           2795
=====

Total params: 169,707
Trainable params: 169,707
Non-trainable params: 0

```

کلاس **Keras Image Data Generator** یک راه سریع و آسان برای تقویت تصاویر ارائه می دهد. این روش مجموعه ای از تکنیک های مختلف مانند استانداردسازی، چرخش، جابجایی، چرخش، تغییر روشنایی و بسیاری موارد دیگر را ارائه می دهد. مزیت اصلی استفاده از کلاس **Keras Image Data Generator** این است که به گونه ای طراحی شده است که افزایش داده ها را در زمان واقعی ارائه دهد. به این معنی که در حالی که مدل شما هنوز در مرحله آموزش است، در حال تولید تصاویر تقویت شده است. کلاس **Image Data Generator** تصمین می کند که مدل تغییرات جدیدی از تصاویر را در هر دوره دریافت می کند. اما فقط تصاویر تبدیل شده را بر می گرداند و آن را به مجموعه اصلی تصاویر اضافه نمی کند. اگر در واقع اینطور بود، پس مدل چندین بار تصاویر اصلی را می دید که قطعاً بیش از حد به مدل ما می رسید. یکی دیگر از مزایای **Image Data Generator** این است که به حافظه کمتری نیاز دارد. این به این دلیل است که بدون استفاده از این کلاس، همه تصاویر را به یکباره بارگذاری می کنیم. اما با استفاده از آن، تصاویر را به صورت دسته ای بارگذاری می کنیم که مقدار زیادی از حافظه را ذخیره می کند.

در این سوال ما از سه نگاشت تصویری:

### **Rotation**

که مربوط به اصلاح زوایه‌ی تصاویر است

### **Shifting**

که مربوط به قرار گیری تابلوهای داده‌ها در محل‌های مختلف تصویر می‌باشد

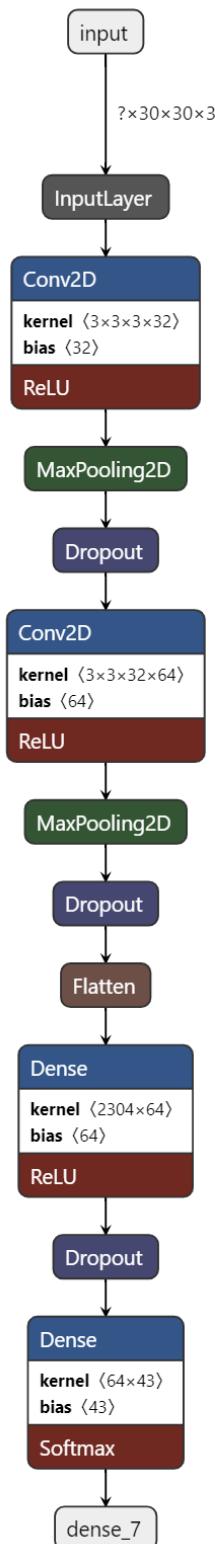
### **Brightness**

که مربوط به اصلاح نور و روشنایی تصاویر می‌باشد

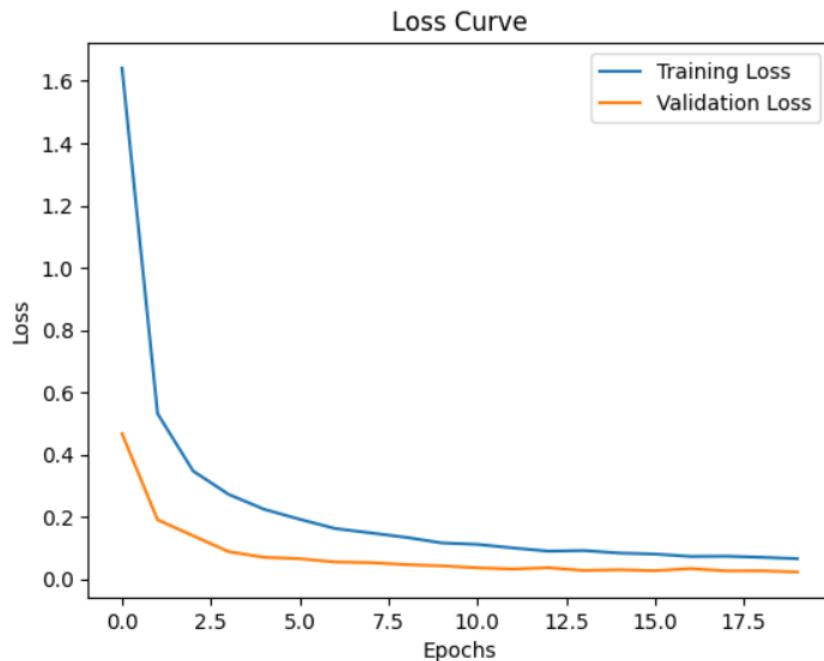
استفاده کرده‌ایم.

علت این موضوع این است که در داده‌های تصویری موجود سه مشکل اصلی مربوط به نور و محل تابلو در عکس و زوایه‌ی تابلو‌ها می‌باشد لذا از این سه نگاشت برای اصلاح همین سه مورد استفاده می‌کنیم.

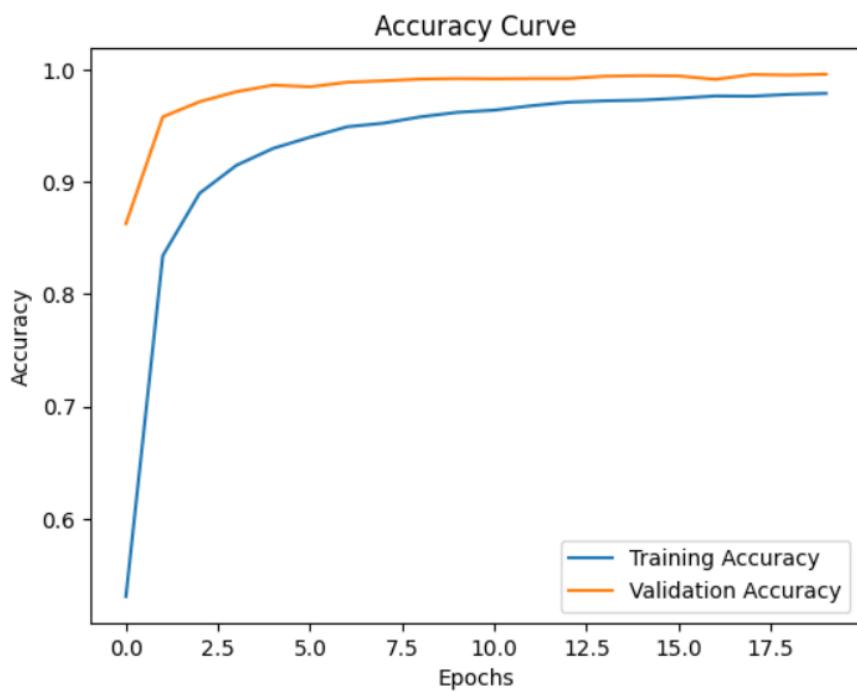
مثلاً چون تصاویری که برعکس گرفته شده باشند بسیار کم است نیازی به استفاده از **flipping** نداریم.



Plot با شبکه پیچشی برای Data Augmentation



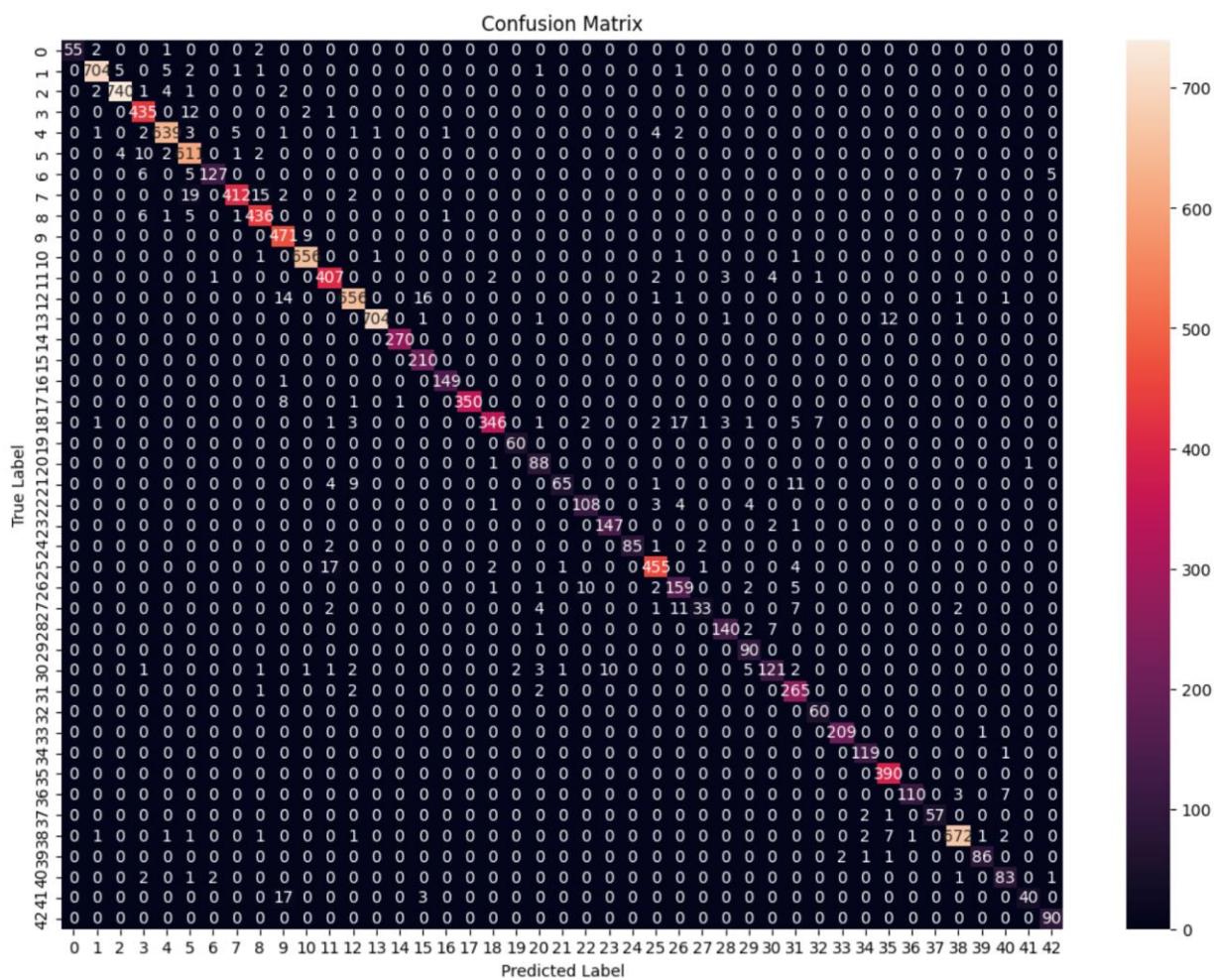
بر حسب تعداد ایپاک **Loss function**



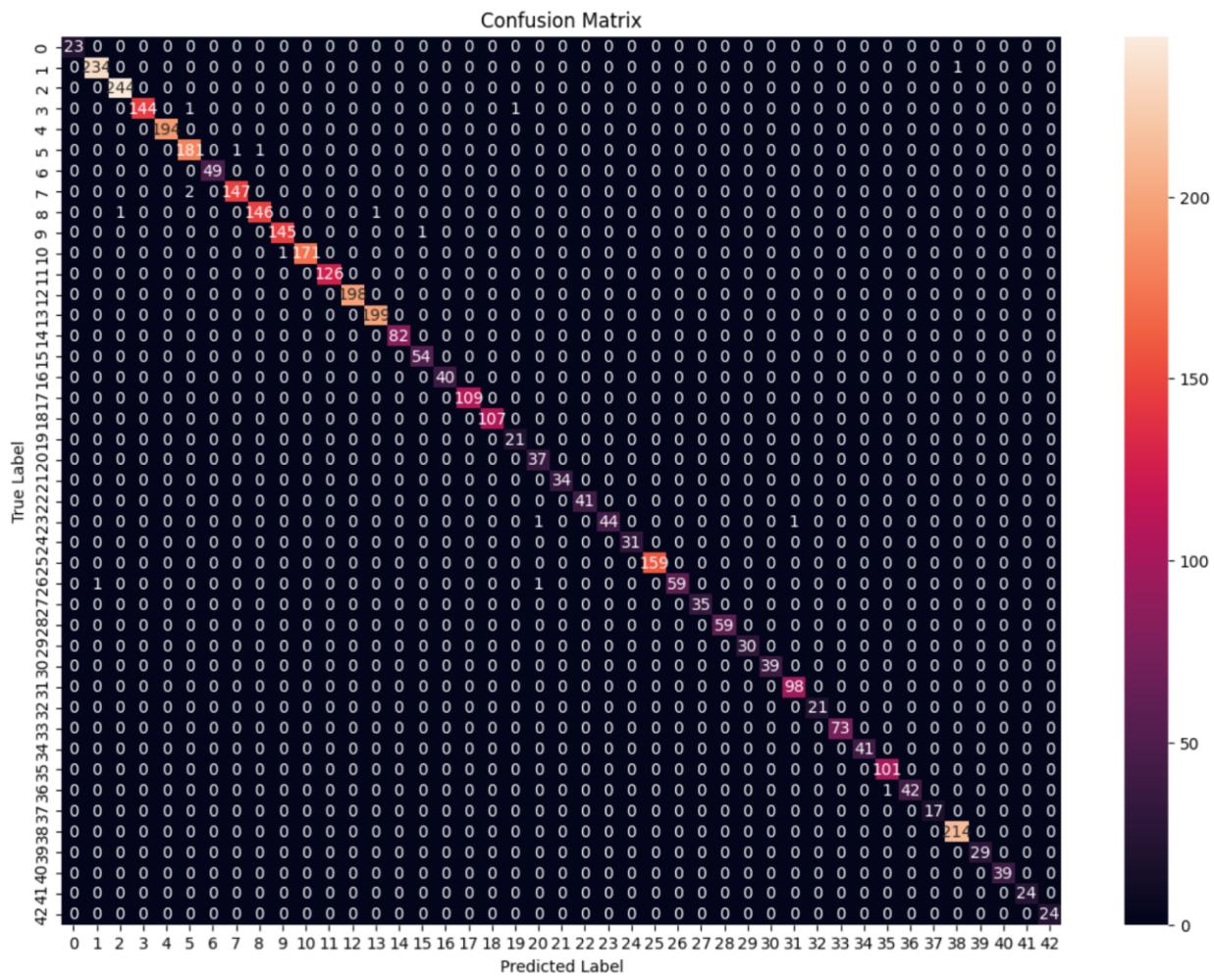
بر حسب تعداد ایپاک **Accuracy**

```
Test Loss: 0.20027558505535126  
Test Accuracy: 0.9588282108306885  
Validation Loss: 0.022990191355347633  
Validation Accuracy: 0.995919406414032
```

## دقت و خطای داده های تست و ارزیابی



## Confusion Matrix برای داده های تست



## Confusion Matrix برای داده های ارزیابی

## مقایسه نتایج

پ) نتایج بخش الف و ب را با نتایج بدست آمده از شبکه ی MLP به اختصار مقایسه کنید.

model	Test Loss	Test Acc	Valid Loss	Valid Acc
MLP	0.752	0.852	0.259	0.94

model	Test Loss	Test Acc	Valid Loss	Valid Acc
CNN Relu -Max pooling Adam- none dropout	0.486	0.938	0.042	0.989

model	Test Loss	Test Acc	Valid Loss	Valid Acc
CNN Relu -Average pooling Adam- none dropout	0.559	0.916	0.066	0.989

model	Test Loss	Test Acc	Valid Loss	Valid Acc
CNN Relu - Max pooling Adam - dropout	0.216	0.958	0.024	0.995

model	Test Loss	Test Acc	Valid Loss	Valid Acc
CNN Relu - Max pooling SGD - dropout	0.389	0.909	0.115	0.976

model	Test Loss	Test Acc	Valid Loss	Valid Acc
CNN Sigmoid - Max pooling SGD - dropout	0.306	0.911	0.048	0.989

model	Test Loss	Test Acc	Valid Loss	Valid Acc
CNN Tan h - Max pooling SGD - dropout	0.186	0.952	0.021	0.993

model	Test Loss	Test Acc	Valid Loss	Valid Acc
CNN Relu - Max pooling Adam - dropout + Data Aug	0.2	0.959	0.023	0.996

نتیجه گیری:

همانطور که در شکل صفحه قبل مشاهده می شود جدول اول مربوط به تربیت شبکه عصبی با روش MLP و سایر روش ها مربوط به تربیت مدل با روش شبکه پیچشی می باشند.

واضح است که همانطور که پیش بینی می شد همه مدل های CNN از مدل MLP دقت بیشتری دارند و خطای کمتری. دقت مدل MLP حدود 85 درصد می باشد در صورتی که کمترین دقت شبکه های پیچشی نیز حدود 90 درصد است.

در میان شبکه های پیچشی اما بهترین مدل مربوط به شبکه ای است که از Max استفاده شده است و تابع بهینه ساز آن Adam می باشد و فعال Saz آن Relu است. همچنین در مدلی با مشخصات مشابه که به جای Relu از Tan h استفاده شده است همگرایی در تعداد ایپاک کمتری اتفاق افتاده است.

اگر بهترین شبکه را به کمک Data Augmentation اصلاح کنیم می بینیم که هم دقت افزایش می یابد و هم تابع زیان یا خطا کاهش می یابد.

لذا بهترین مدل مربوط به شبکه ای است که از Dropout و Max pooling استفاده شده است و تابع بهینه ساز آن Adam می باشد و فعال ساز آن Relu است و در آن از Data Augmentation استفاده شده است.

همانطور که در جدول آخر مشاهده می شود دقت داده های تست این مدل 95.9 درصد و تابع زیان آن 0.2 می باشد که در بین همه شبکه های تربیت شده بهترین نتیجه است.

منابع:

- ١ ويکی پدیا
- ٢ <https://www.w3schools.com>
- ٣ Youtub.com
- ٤ [https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.train\\_test\\_split.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)
- ٥ Chat GPT
- ٦ <https://www.analyticsvidhya.com>
- ٧ Pypi.org