



به نام خدا



دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر
درس شبکه‌های عصبی و یادگیری عمیق

پروژه شماره 1

نام و نام خانوادگی	محسن فیاض - محمد رضا عظیمی
شماره دانشجویی	810100521 - 810100524
تاریخ ارسال گزارش	شنبه، 3 اردیبهشت 1401

فهرست گزارش سوالات

1

سوال 1 – CNN - Classification

(الف) در این بخش تنها با استفاده از لایه های کانولوشنی در بخش استخراج ویژگی، شبکه را پیاده سازی کنید. دقت و خطای شبکه را پس از آموزش، گزارش کنید.

(ب) لایه های Pooling و Batch normalization را توضیح دهید سپس این لایه ها را به توپولوژی شبکه اضافه و شبکه را پیاده سازی کنید. نتایج بدست آمده از نظر دقت خطا با معماری قسمت(الف) مقایسه کنید.

(ج) به معماری شبکه بدست آمده در قسمت(ب) اکنون dropout رانیز اضافه و تاثیر آن را بررسی کنید. چرا از dropout در معماری شبکه عصبی استفاده می کنیم؟

(د) توقف زود هنگام شبکه های عصبی به چه معناست؟ چه معیارهایی در این توقف زود هنگام استفاده میشوند؟ یک نمونه از آن پیاده سازی کنید.

11

سوال 2 – Transfer Learning

(الف) موارد زیر را در مورد مدل انتخابی خود توضیح دهید: معماری شبکه، مزایا و معایب، در صورتی که تصویر ورودی شبکه به پیش پردازش اولیه ای نیاز دارد آن را توضیح دهید.

(ب) تعریفی از transfer learning ارائه دهید و شبکه انتخابی را به کمک transfer learning پیاده سازی کنید و قسمت ج را اجرا کنید.

(ج) ابتدا مشخص کنید که چه دسته عکسهایی توسط مدل انتخابی شما قابل تشخیص هستند. سپس یک عکس دلخواه از دسته های قابل تشخیص را انتخاب کنید. در صورت لزوم پیش پردازش های لازم را بر روی عکس انجام دهید و به شبکه ای که در بخش قبل طراحی کرده اید دهید 3 دسته به ترتیب با بیشترین احتمال پیش بینی شده را در خروجی نشان دهید.

(د) یک دسته از عکس های مختلفی که توسط مدل شما قابل تشخیص باشد را به دلخواه انتخاب نمایید، پیش پردازش های لازم را انجام دهید و مدل را دوباره آموزش دهید. (می توانید از دیتابست های معروفی همچون imageNet استفاده نمایید و یا دیتابست های دیگری که به نظر مناسب می آیند)

19

سوال 3 – Segmentation

24

سوال 4 – Object Detection

24

(الف)

27

(ب)

28

(پ)

29

(ت)

36

بازخورد

CNN - Classification – 1 سوال

کلاس‌های موجود در این دیتاست شامل موارد زیر است.

جدول 1 - کلاس‌های موجود در دیتاست

dog	5	airplane	0
frog	6	automobile	1
horse	7	bird	2
ship	8	cat	3
truck	9	deer	4

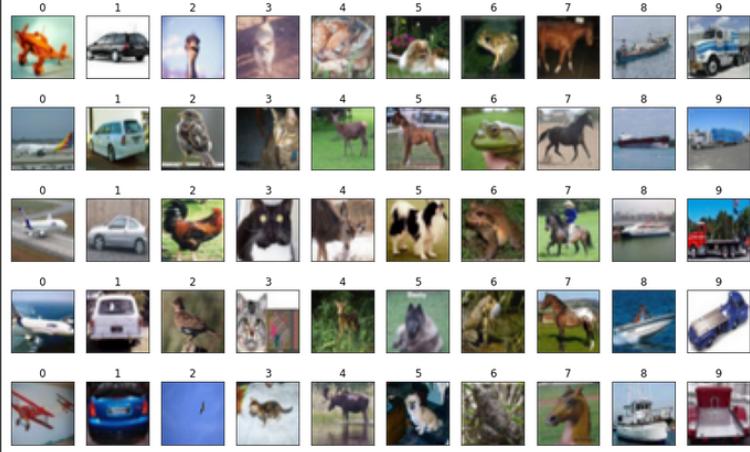
ابتدا دیتاست را بارگذاری کرده و 10 تصویر آن را به صورت تصادفی نمایش می‌دهیم. (از هر کلاس یکی و 5 بار این کار را تکرار کردیم تا بهتر کلاس‌ها مشخص شوند.)

```
1 from keras.datasets import cifar10
2 import tensorflow as tf
3 import matplotlib.pyplot as plt
4 import numpy as np

1 [x_train, y_train], [x_test, y_test] = cifar10.load_data()
2 print("x_train:", x_train.shape)
3 print("y_train:", y_train.shape)

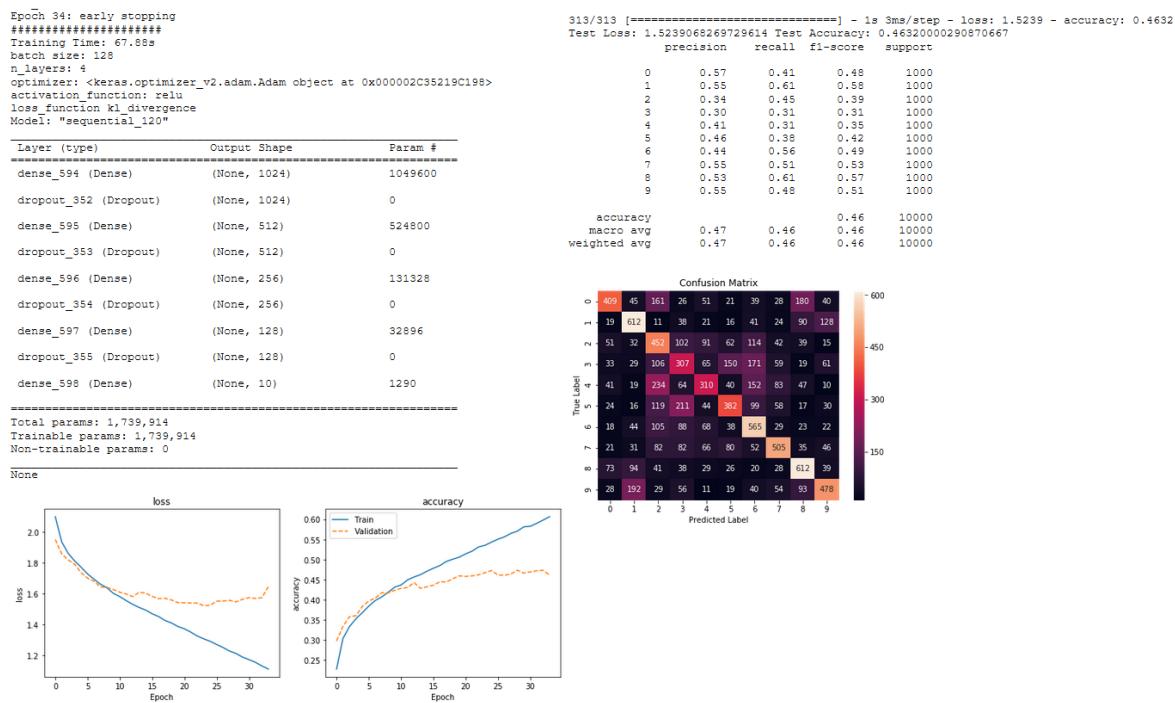
x_train: (50000, 32, 32, 3)
y_train: (50000, 1) <class 'numpy.ndarray'>

1 def plot_pics(x: np.ndarray, y: np.ndarray):
2     fig = plt.figure(figsize=(15, 4))
3     for class_num in range(10):
4         class_indices = np.where(y == class_num)[0]
5         class_x = x[class_indices]
6         rnd_idx = np.random.randint(len(class_x))
7         image = class_x[rnd_idx]
8         ax = fig.add_subplot(1, 10, class_num + 1, xticks=[], yticks[])
9         ax.set_title(class_num)
10        plt.imshow(image)
11    plt.show()
12
13 for i in range(5):
14     plot_pics(x_train, y_train)
```



شکل 1 - بارگذاری و نمایش دیتاست

ساختار شبکه، نمودار خطای و دقت در طول آموزش روی داده آموزش و ارزیابی که در تمرین قبلی استفاده شده بود در زیر آمده است.



شکل 2 - نتیجه بهترین مدل

نتیجه تمرین قبلی:

● خطای تست: 1.5239

● دقت تست: 0.4632

● زمان آموزش: 67.88s در 34 اپاک

در نمودار ها هم دیده می شود که جایی که دیگر لاس ارزیابی در حال بالا رفتن است (overfitting) دیگر متوقف می شویم تا generalization مدل را از دست ندهیم. همچنین باید دقت داشت که چون 10 کلاس داریم، دقت مدل رندوم 10 درصد خواهد بود که دقت 46 این مدل خوب است. البته با استفاده از CNN ها می توان دقت های خیلی بیشتر گرفت که در بخش های این سوال به پیاده سازی آن می پردازیم.

الف) در این بخش تنها با استفاده از کیه های کانولوشنی در بخش استخراج ویژگی، شبکه را پیاده سازی کنید. دقت و خطای شبکه را پس از آموزش، گزارش کنید.

کد پیاده سازی شده در زیر آمده است.

```

class Trainer:
    def __init__(self, activation_function="relu",
                 optimizer="adam", loss_function='categorical_crossentropy',
                 pooling=False, batch_norm=False, drop_out=False) -> None:
        self.optimizer = optimizer
        self.activation_function = activation_function
        self.loss_function = loss_function
        self.pooling = pooling
        self.batch_norm = batch_norm
        self.drop_out = drop_out
        self.model = self.build_cnn_model()
        self.history = None
        self.training_time = None

    def print_summary(self):
        print("#####")
        print(f"Training Time: {self.training_time:.2f}s")
        print("batch size:", self.batch_size)
        print("optimizer:", self.optimizer)
        print("activation_function:", self.activation_function)
        print("loss_function", self.loss_function)
        print(self.model.summary())

    def build_cnn_model(self):
        def add_max_pool(model):
            if self.pooling:
                model.add(tf.keras.layers.MaxPooling2D((2, 2)))

        def add_batch_normalization(model):
            if self.batch_norm:
                model.add(tf.keras.layers.normalization.BatchNormalization())

        def add_drop_out(model):
            if self.drop_out:
                model.add(tf.keras.layers.Dropout(0.2))

        model = tf.keras.models.Sequential()
        model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3)))
        add_max_pool(model)
        add_batch_normalization(model)
        model.add(tf.keras.layers.Activation(self.activation_function))
        add_drop_out(model)

        model.add(tf.keras.layers.Conv2D(64, (3, 3)))
        add_max_pool(model)
        add_batch_normalization(model)
        model.add(tf.keras.layers.Activation(self.activation_function))
        add_drop_out(model)

        model.add(tf.keras.layers.Conv2D(64, (3, 3)))
        add_max_pool(model)
        add_batch_normalization(model)
        model.add(tf.keras.layers.Activation(self.activation_function))
        add_drop_out(model)
        model.add(tf.keras.layers.Flatten())
        model.add(tf.keras.layers.Dense(64, activation='relu'))
        model.add(tf.keras.layers.Dense(10, activation='softmax'))

        model.compile(
            optimizer=self.optimizer,
            loss=self.loss_function,
            metrics=[accuracy]
        )
        return model

    def train(self, x_train, y_train, batch_size=32, epochs=50):

```

```

        self.batch_size = batch_size
        assert y_train.shape[1:] == (10, )
        early_stopping = tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            verbose=1,
            patience=4,
            mode='min',
            restore_best_weights=True
        )
        start = time()
        self.history = self.model.fit(
            x_train, y_train,
            batch_size=batch_size,
            epochs=epochs,
            verbose=1,
            validation_split=0.2,
            callbacks=[early_stopping],
        )
        self.training_time = time() - start

    def plot_history(self):
        fig = plt.figure(figsize=(12, 4))
        metrics = ['loss', 'accuracy']
        for n, metric in enumerate(metrics):
            plt.subplot(1, 2, n+1)
            plt.plot(self.history.epoch, self.history.history[metric], label='Train')
            plt.plot(self.history.epoch, self.history.history[f"val_{metric}"], linestyle="--", label='Validation')
            plt.xlabel('Epoch')
            plt.ylabel(metric)
            plt.title(metric)
        plt.legend()
        plt.show()

    def evaluate(self, x_test, y_test):
        assert y_test.shape[1:] == (10, )
        [test_loss,test_acc] = self.model.evaluate(x_test,y_test)
        print("Test Loss:", test_loss, "Test Accuracy:", test_acc)
        test_preds = np.argmax(self.model.predict(x_test), axis=-1)
        y_test = np.argmax(y_test, axis=-1)
        print(classification_report(y_test, test_preds))
        self.plot_cm(y_test, test_preds)

    def plot_cm(self, y_true, preds):
        cm = confusion_matrix(y_true, preds)
        plt.figure(figsize=(7, 5))
        ax = sns.heatmap(cm, annot=True, fmt="d")
        bottom, top = ax.get_ylim()
        ax.set_ylim(bottom + 0.5, top - 0.5)
        plt.title('Confusion Matrix')
        plt.ylabel('True Label')
        plt.xlabel('Predicted Label')
        plt.show()

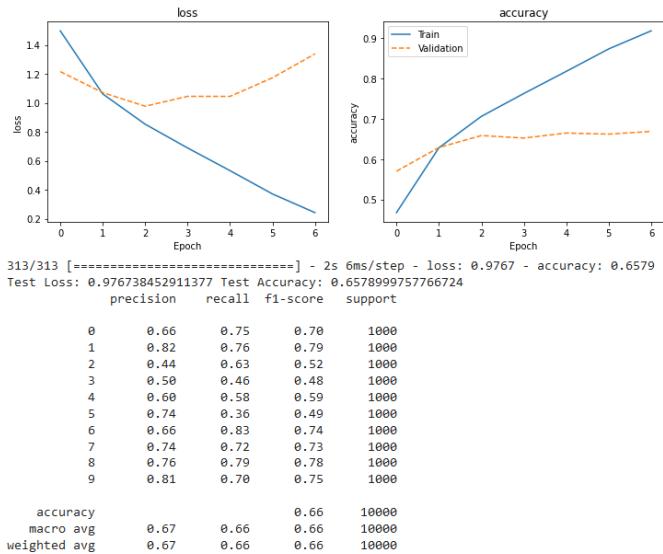
```

کد 1 - پایه سازی مسئله با استفاده از CNN ها

در این بخش همانطور که خواسته شده تنها با استفاده از لایه های کانولوشنی برای استخراج ویژگی ها مدل را آموزش می دهیم و نتایج آن را گزارش می کنیم.

```

Epoch 7: early stopping
#####
Training Time: 71.40s
batch size: 128
optimizer: <keras.optimizer_v2.adam.Adam object at 0x7f46e0397d10>
activation_function: relu
loss_function kl_divergence
Model: "sequential_3"
Layer (type)          Output Shape         Param #
conv2d_9 (Conv2D)     (None, 30, 30, 32)   896
activation_9 (Activation) (None, 30, 30, 32)   0
conv2d_10 (Conv2D)    (None, 28, 28, 64)    18496
activation_10 (Activation) (None, 28, 28, 64)   0
conv2d_11 (Conv2D)    (None, 26, 26, 64)    36928
activation_11 (Activation) (None, 26, 26, 64)   0
flatten_3 (Flatten)   (None, 43264)        0
dense_6 (Dense)       (None, 64)           2768960
dense_7 (Dense)       (None, 10)           650
=====
Total params: 2,825,930
Trainable params: 2,825,930
Non-trainable params: 0
=====
```



شکل 3 - نتیجه استفاده از CNN ها

نتیجه استفاده تنها از CNN:

- خطای خطا: 0.9767
- دقت: 0.6579
- زمان کل آموزش: 71.40s در 7 اپیک

همانطور که در درس هم توضیح داده شده بود، استفاده از CNN ها باعث شده است که مرحله به مرحله ویژگی های مناسب در تصاویر تشخیص داده شوند و بازنمایی مناسبی از تصویر ساخته شود تا در نهایت در یک فضای ویژگی که به خوبی جدایی پذیر است یک شبکه ساده بتواند کلاس ها را تشخیص دهد. دقت 65 درصدی نسبت به تمرین قبلی 46 درصد بود کاملا نشان دهنده این موضوع است.

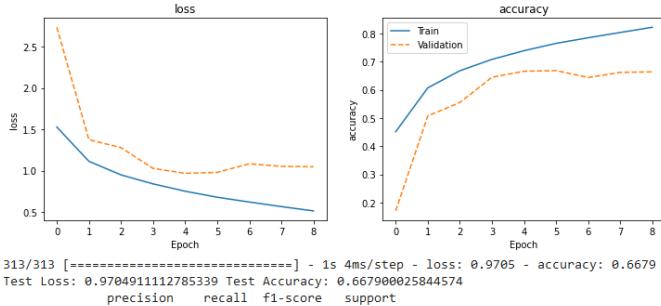
ب) لایه های Pooling و Batch normalization را توضیح دهید سپس این لایه ها را به توپولوژی شبکه اضافه و شبکه را پیاده سازی کنید. نتایج بدست آمده از نظر دقیقت خطا با معما ری قسمت (الف) مقایسه کنید.

لایه Pooling که معمولاً به شکل max یا avg پولینگ است یک پنجره را حرکت می‌دهد و مثلاً در max pooling مقدار ماکسیمم آن پنجره را جایزگین کل پنجره می‌کند. این کار مانند مثالی که در درس آورده شد می‌تواند از distortion ها بکاهد و robustness را افزایش دهد. مثل مثال درس که یک دایره اگرچه کمی از اطراف کشیده شده باشد در نهایت با pooling می‌تواند شبیه یک دایره درست شود. در مورد عکس مثلاً ممکن است چشم در یک مکان حدودی با جابجایی نویزی دیده شود ولی پولینگ مکان را یکسانسازی و از نویز می‌کاهد و مهمترین ویژگی هایی که پیدا شده اند را فقط نگه میدارد. البته قطعاً اینها توضیحات و توجیه هستند و بررسی دقیق نحوه کار آن نیازمند تحقیق و بررسی است.

نرمال سازی دسته ای یا internal covariate shift یک مشکل عمدۀ به نام batch normalization یکی است که میانی شبکه عصبی shift را حل می‌کند. این روش کمک می‌کند تا داده ها بین لایه های میانی شبکه عصبی نرمال شوند. این کار تاثیر Regularization دارد که کمک می‌کند هم مسئله راحت تر همگرا شود (همگرایی دیتای نرمال راحت تر از دیتایی است که نرمال نباشد) و همچنین در نهایت باعث generalization بهتر می‌شود.

در این بخش این دو مورد را اضافه می‌کنیم و مدل را دوباره آموزش می‌دهیم تا تاثیر این دو را مشاهده کنیم.

Layer (type)	Output Shape	Param #
conv2d_16 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_4 (MaxPooling 2D)	(None, 15, 15, 32)	0
batch_normalization_3 (Batch Normalization)	(None, 15, 15, 32)	128
activation_15 (Activation)	(None, 15, 15, 32)	0
conv2d_17 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_5 (MaxPooling 2D)	(None, 6, 6, 64)	0
batch_normalization_4 (Batch Normalization)	(None, 6, 6, 64)	256
activation_16 (Activation)	(None, 6, 6, 64)	0
conv2d_18 (Conv2D)	(None, 4, 4, 64)	36928
max_pooling2d_6 (MaxPooling 2D)	(None, 2, 2, 64)	0
batch_normalization_5 (Batch Normalization)	(None, 2, 2, 64)	256
activation_17 (Activation)	(None, 2, 2, 64)	0
flatten_5 (Flatten)	(None, 256)	0
dense_10 (Dense)	(None, 64)	16448
dense_11 (Dense)	(None, 10)	650
<hr/>		
Total params: 74,058	Epoch 9: early stopping	●
Trainable params: 73,738	#####	●
Non-trainable params: 320	Training Time: 38.45s	●



313/313 [=====] - 1s 4ms/step - loss: 0.9705 - accuracy: 0.6679
Test Loss: 0.970491111278539 Test Accuracy: 0.667900025844574

precision recall f1-score support

	0	0.63	0.78	0.70	1000
1	0.76	0.85	0.80	1000	
2	0.56	0.54	0.55	1000	
3	0.51	0.49	0.50	1000	
4	0.70	0.50	0.59	1000	
5	0.60	0.61	0.60	1000	
6	0.76	0.74	0.75	1000	
7	0.64	0.74	0.69	1000	
8	0.88	0.62	0.73	1000	
9	0.70	0.81	0.75	1000	

	accuracy				
macro avg	0.67	0.67	0.67	10000	
weighted avg	0.67	0.67	0.67	10000	

شکل 4 - نتیجه استفاده از CNN و Pooling و batch normalization

نتیجه اجرا با استفاده از CNN و Pooling و batch normalization :

خطا: 0.9705 ●

دقت: 0.6679 ●

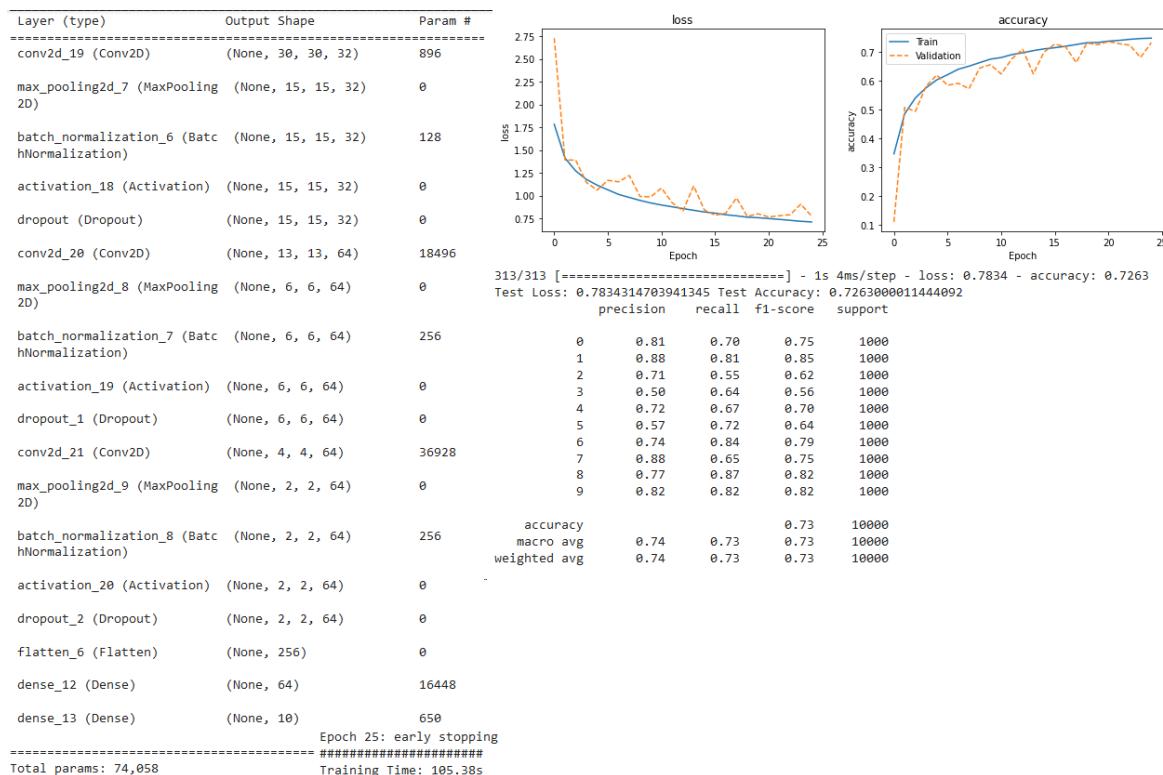
زمان کل آموزش: 38.45s در 9 ایپاک ●

همانطور که دیده می شود و طبق توضیحاتی که داده شد، اضافه کردن این دو مورد تاثیر مثبت داشته و خطأ و دقت نسبت به حالت قبلی به ترتیب کمتر و بیشتر شدند که مناسب است.

ج) به معنایی شبکه بدست آمده در قسمت (ب) اکنون dropout، اینز اضافه و تاثیر آن را بررسی کنید. چراز dropout د، معنایی شبکه عصبی استفاده می کنیم؟

اضافه کردن لایه dropout یک روش regularization است که باعث تعمیم پذیری و generalization overfitting از جلوگیری از می شود. این روش با خاموش کردن درصدی از نورون ها به صورت رندوم در طول آموزش باعث می شود که مدل بیش از حد به نتیجه یک نورون وابسته نشود و overfit نکند. لازم به ذکر است که این کار در زمان تست دیگر انجام نمی شود و این لایه در تست بی اثر می شود.

در زیر نتیجه اضافه کردن این لایه آمده است.



شکل 5 - نتیجه استفاده از CNN و Pooling و batch normalization و dropout

نتیجه استفاده از CNN و Pooling و batch normalization و dropout :

خطا: 0.7834 ●

دقت: 0.7263 ●

زمان کل آموزش: 105.38s در 25 اپیاک ●

همانطور که دیده می شود اضافه کردن dropout با خاصیت regularization که داشت باعث شد مدل خیلی بهتر آموزش ببیند (هر چند کمی بیشتر طول کشید) و خطأ به طور چشمگیری

کاهش پیدا کرد و دقت به طور قابل توجهی افزایش پیدا کرد. علت این مورد در بالاتر توضیح داده شد.

د) توقف زود هنگام شبکه های عصبی به چه معناست؟ چه معیارهایی در این توقف زود هنگام استفاده می‌شوند؟ یک نمونه از آن پیاده سازی کنید.

توقف زود هنگام یا early stopping یعنی قبل از آنکه آموزش مدل به سمت overfitting برود که یعنی روی داده آموزش بهتر و بهتر شود ولی با هزینه اینکه generalization خود را از دست دهد و روی داده validation ضعیف تر عمل کند، آموزش را متوقف کنیم. برای این کار اکثرا از معیار مقدار loss استفاده می‌شود به این صورت که مثلا اگر loss دیتابی در 5 اپیاک کاهش نیافته بود یعنی به overfitting رسیدیم و باید متوقف شویم. حتی می‌توان پس از توقف به جای آخری اپیاک، مدلی که کمترین loss روی validation را استفاده کنیم. البته می‌توان از معیارهای دیگری مثل f1 score یا accuracy یا validation محاسبه شود استفاده کرد. فقط باید توجه داشت که مینیمم یا ماکسیمم آن معیار برای توقف اهمیت دارد. و همچنین باید توجه داشت که همیشه معیار توقف، کیفیت مدل روی داده validation است نه داده آموزش.

در تمام بخش‌های قبلی از early stopping برای توقف و داشتن مقایسه عادلانه تر استفاده شده بود. برای این کار از تکه کد زیر استفاده کردیم.

```
early_stopping = tf.keras.callbacks.EarlyStopping(  
    monitor='val_loss',  
    verbose=1,  
    patience=4,  
    mode='min',  
    restore_best_weights=True  
)
```

کد 2 - پیاده سازی توقف زودهنگام

در این مورد از مقدار loss validation برای توقف استفاده کردیم، مقدار صبوری را 4 اپیاک قرار دادیم (بعد از 4 اپیاک عدم کاهش لاس متوقف شویم) و چون مقدار لاس کمینه اش مناسب است نوع را از نوع مینیمم گذاشتیم و همچنین در آخر آموزش مدلی را استفاده می‌کنیم که بهترین مقدار loss validation را داشته نه لزوما آخرین اپیاک اجرا شده.

از همین حالت در نتایج بخش ج استفاده شده است که برای نمودار لاس آموزش و validation به ج رجوع شود. دیده می‌شود که لاس آموزش رو به کاهش است اما لاس ارزیابی دارد به سمت مسطح شد و سپس بالا رفتن می‌رود که دیگر early stopping در اپیک 25 آن را متوقف کرده است. برای سرعت بیشتر صبوری 4 گذاشته شده بود، اگر بیشتر گذاشته شود بالا رفتن لاس ارزیابی را با وضوح بهتری هم می‌توان دید.

سوال ۲ Transfer Learning – ۲

مجموع ۴ و ۱ برابر ۵ است و در این سوال به بررسی ENet می‌پردازیم.

الف) موارد زیر را در مورد مدل انتخابی خود توضیح دهید: معماری شبکه، مزایا و معایب، در صورتی که تصویر ورودی شبکه به پیش‌پردازش اولیه ای نیاز دارد آن را توضیح دهید.

شبکه ENet یا همان EfficientNet در مقاله‌ای از Quoc V. Le Mingxing Tan و معرفی شده است^۱. ۸ حالت مختلف از این مدل وجود دارد که سایزهای ورودی متنوعی دارد. وزنهایی که استفاده می‌کنیم از آموزش روی imangenet خواهد بود که در آخر 1000 کلاسه است و بخش بالایی را نیز بارگذاری می‌کنیم.

در معماری این مدل به جای روش‌های معمول scale اینکار را بهتر انجام می‌دهد. این scale می‌تواند در جهت عمق، رزولوشن، یا پهنا (تعداد فیچرمپ) باشد. در این روش هر ۳ کار با هم انجام می‌شوند. در مقاله این کار نشان داده می‌شود که این کار بهتر از انجام هر کدام به تنها یکی است. از مزایای این کار در مقاله آمده است که می‌توان با شبکه‌های کوچکتر به این شکل نتایج بسیار خوبی گرفت بنابراین سرعت این مدل نسبت به مدل‌های قبلیش بهتر بوده است. ایده پشت این کار این است که اگر رزولوشن عکس بزرگتر باشد، باید پهنا که فیچرمپ‌ها هستند نیز بزرگتر شوند تا فیچرهای بیشتری را بگیرند و همچنین باید عمق بیشتر شود تا بازنمایی‌های دقیق‌تری از تصویر ساخته شود. این روش که آنرا compound scaling می‌نامند در مقاله نشان داده شده روی هر مدل دیگری نیز می‌تواند پیاده شود و اگر با این روش عمق و پهناش مشخص شود می‌تواند نتایج بهتری بگیرد. از معایب این کار می‌توان این را دانست که برای پیدا کردن بهترین هایپرپارامترها برای این grid scaling استفاده می‌شود و در مقاله نیز گفته شده که هر چه مدل بزرگتر باشد این کار نیز هزینه برتر خواهد بود. همچنین می‌توان محدودیت کرنل سایز در هر لایه را نیز مشکل این مدل دانست که در inception حل می‌شود.

برای پیش‌پردازش نیازی نیست اعداد نمایش شوند و باید بین ۰ و 255 باشند. البته سایز ورودی باید به سایز مناسب تغییر کند. این مدل حالت B0 تا B7 دارد که از سایز 224 تا 600 است.

^۱ <https://arxiv.org/pdf/1905.11946.pdf>

برای سبکی محاسبات از مدل B0 استفاده می‌کنیم. بنابراین یک لایه به ابتدای مدل اضافه کردیم که سایز ورودی را به 224 تبدیل کند.

معماری نهایی مدل به این شکل است.

Layer (type)	Output Shape	Param #	Connected to
<hr/>			
input_1 (InputLayer)	[(None, 32, 32, 3)]	0	[]
resizing (Resizing)	(None, 224, 224, 3)	0	['input_1[0][0]']
rescaling (Rescaling)	(None, 224, 224, 3)	0	['resizing[0][0]']
normalization (Normalization)	(None, 224, 224, 3)	7	['rescaling[0][0]']
stem_conv_pad (ZeroPadding2D)	(None, 225, 225, 3)	0	['normalization[0][0]']
stem_conv (Conv2D)	(None, 112, 112, 32)	864	['stem_conv_pad[0][0]']
stem_bn (BatchNormalization)	(None, 112, 112, 32)	128	['stem_conv[0][0]']
stem_activation (Activation)	(None, 112, 112, 32)	0	['stem_bn[0][0]']
block1a_dwconv (DepthwiseConv2D)	(None, 112, 112, 32)	288	['stem_activation[0][0]']
D))
block1a_bn (BatchNormalization)	(None, 112, 112, 32)	128	['block1a_dwconv[0][0]']
block1a_activation (Activation)	(None, 112, 112, 32)	0	['block1a_bn[0][0]']
...			
...			
block7a_se_reshape (Reshape)	(None, 1, 1, 1152)	0	['block7a_se_squeeze[0][0]']
block7a_se_reduce (Conv2D)	(None, 1, 1, 48)	55344	['block7a_se_reshape[0][0]']
block7a_se_expand (Conv2D)	(None, 1, 1, 1152)	56448	['block7a_se_reduce[0][0]']
block7a_se_excite (Multiply)	(None, 7, 7, 1152)	0	['block7a_activation[0][0]', 'block7a_se_expand[0][0]']
block7a_project_conv (Conv2D)	(None, 7, 7, 320)	368640	['block7a_se_excite[0][0]']
block7a_project_bn (BatchNormalization)	(None, 7, 7, 320)	1280	['block7a_project_conv[0][0]']
top_conv (Conv2D)	(None, 7, 7, 1280)	409600	['block7a_project_bn[0][0]']
top_bn (BatchNormalization)	(None, 7, 7, 1280)	5120	['top_conv[0][0]']
top_activation (Activation)	(None, 7, 7, 1280)	0	['top_bn[0][0]']
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0	['top_activation[0][0]']
dense (Dense)	(None, 10)	12810	['global_average_pooling2d[0][0]']
<hr/>			
Total params: 4,062,381			
Trainable params: 4,020,358			
Non-trainable params: 42,023			

برای اطمینان توضیحات این بخش را برای مدل inception هم تکرار می‌کیم. در معماری این مدل از کرنل‌های با سایزهای مختلف استفاده شده است تا نیاز نباشد سایز کرنل به عنوان هایپرپارامتر مسئله باشد تا ما تنظیمش کنیم. بلکه خود مدل با این کار بهترین فیچرها را در هر لایه با هر سایز کرنلی که بخواهد به دست می‌آورد. از موارد مفید این کار همین موضوع که توضیح داده شد است که مدل خودش بهترین سایزهای کرنل را استفاده می‌کند، اما از معایب آن این است که محاسبات بیشتری نسبت به مدل عادی خواهد داشت و باید چندین کرنل به جای یکی در هر لایه آموزش ببینند که آموزش را کنترل خواهد کرد.

ب) تعریف از **transfer learning** ائه دهید و شبکه انتخابی را به کمک **learning** پیاده سازی کنید و قسمت ج را اجرا کنید.

روش transfer learning یک روش یادگیری ماشین است که در آن از یک مدل پیشآموزش داده شده به عنوان نقطه شروع یک مدل جدید استفاده می‌کنیم. این کار باعث می‌شود از آموزش مدل اولیه بهره ببریم و سریعتر به مدلی که نیاز داریم همگرا شویم. مثلاً در مسائل ویژن وقتنی از مدلی که قبلاً روی imagenet آموزش دیده است استفاده می‌شود در اصل آموزش‌های اولیه مثل تشخیص لبه‌ها و حتی موارد پیچیده‌تر مثل خودرو استفاده می‌کنیم و سپس اگر مثلاً بخواهیم مدلی بسازیم که به جز خودرو بودن مدل خودرو را نیز تشخیص دهد، نیازی نیست آموزش یادگیری لبه‌ها و اجزای اصلی خودرو را دوباره یاد بگیریم و خیلی در آموزش جلو میافتیم. مشابه این مورد در کاربردهای NLP نیز وجود دارد که مدل‌های پیشآموزش دیده مانند BERT با pre training objective مانند masked language می‌توان با سرعت خیلی بیشتر finetune می‌بینند و سپس آنها را برای تسک‌های downstream می‌توان با سرعت خیلی بیشتر کرد و به نتیجه مناسب رسید چون کلیت دانش زبانی transfer شده است و فقط تغییرات جزئی برای آموزش تسک خاص ما لازم است.

کدی که برای transfer learning پیاده سازی شده است در ادامه آمده است.

```
class Trainer:
    def __init__(self, activation_function="relu",
                 optimizer="adam",
                 loss_function='categorical_crossentropy') -> None:
        self.optimizer = optimizer
        self.activation_function = activation_function
        self.loss_function = loss_function
        self.model = self.build_cnn_model()
        self.history = None
        self.training_time = None

    def print_summary(self):
        print("#####")
```

```

print(f"Training Time: {self.training_time:.2f}s")
print("batch size:", self.batch_size)
print("optimizer:", self.optimizer)
print("activation_function:", self.activation_function)
print("loss_function", self.loss_function)
print(self.model.summary())

def build_cnn_model(self):
    model = tf.keras.applications.EfficientNetB0(
        include_top=False,
        weights="imagenet",
        input_tensor=tf.keras.layers.Resizing(224, 224)(tf.keras.Input(shape=(32, 32, 3)))
    )
    x = model.layers[-1].output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(10, activation='softmax')(x)
    model = tf.keras.Model(inputs=model.inputs, outputs=x)

    model.compile(
        optimizer=self.optimizer,
        loss=self.loss_function,
        metrics=['accuracy']
    )
    return model

def train(self, x_train, y_train, batch_size=32, epochs=50):
    self.batch_size = batch_size
    assert y_train.shape[1:] == (10, )
    early_stopping = tf.keras.callbacks.EarlyStopping(
        monitor='val_loss',
        verbose=1,
        patience=4,
        mode='min',
        restore_best_weights=True
    )
    start = time()
    self.history = self.model.fit(
        x_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        verbose=1,
        validation_split=0.2,
        callbacks=[early_stopping],
    )
    self.training_time = time() - start

def plot_history(self):
    fig = plt.figure(figsize=(12, 4))
    metrics = ['loss', 'accuracy']
    for n, metric in enumerate(metrics):
        plt.subplot(1, 2, n+1)
        plt.plot(self.history.epoch, self.history.history[metric], label='Train')
        plt.plot(self.history.epoch, self.history.history[f"val_{metric}"], linestyle="--", label='Validation')
        plt.xlabel('Epoch')
        plt.ylabel(metric)
        plt.title(metric)
    plt.legend()
    plt.show()

def evaluate(self, x_test, y_test):
    assert y_test.shape[1:] == (10, )
    [test_loss, test_acc] = self.model.evaluate(x_test, y_test)
    print("Test Loss:", test_loss, "Test Accuracy:", test_acc)
    test_preds = np.argmax(self.model.predict(x_test), axis=-1)
    y_test = np.argmax(y_test, axis=-1)
    print(classification_report(y_test, test_preds))
    self.plot_cm(y_test, test_preds)

def plot_cm(self, y_true, preds):
    cm = confusion_matrix(y_true, preds)
    plt.figure(figsize=(7, 5))
    ax = sns.heatmap(cm, annot=True, fmt="d")
    bottom, top = ax.get_ylim()
    ax.set_ylim(bottom + 0.5, top - 0.5)
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

```

کتاب پاده سازی - ۳ transfer learning

ج) ابتدا مشخص کنید که چه دسته عکس‌هایی توسط مدل انتخابی شما قابل تشخیص هستند. سپس یک عکس دلخواه از دسته های قابل تشخیص را انتخاب کنید. در صورت لزوم پیش پردازش های لازم را بر روی عکس انجام دهید و به شبکه ای که در بخش قبل طراحی کرده اید دهید 3 دسته به ترتیب با بیشترین احتمال پیش بینی شده را در خروجی نشان دهید.

وزنهای مدل برای آموزش روی داده imangenet بارگذاری شده اند که 1000 کلاس دارد.

بعضی از کلاسهای آن برای نمونه هستند 8 hen و 21 kite و 954 banana و 928 ice cream. تمام کلاس در این [لنک²](#) موجود است.

برای آزمون مدل از عکس زیر (از اینترنت) استفاده می‌کنیم که باید در دسته 928 قرار گیرد.



شکل 6 - عکس برای تست مدل

کد، پیش پردازش لازم (تغییر ابعاد و اضافه کردن بعد batch)، دانلود مدل، اجرای آن و سورت کردن نتایج در زیر آمده است.

² <https://deeplearning.cms.waikato.ac.nz/user-guide/class-maps/IMAGENET/>

```

1 model = tf.keras.applications.EfficientNetB0(
2     include_top=True,
3     weights="imagenet",
4     input_tensor=None,
5     input_shape=None,
6     pooling=None,
7     classes=1000,
8     classifier_activation="softmax",
9 )

```

□ Downloading data from <https://storage.googleapis.com/keras-applications/efficientnetb0.h5>
21839872/21834768 [=====] - 0s 0us/step
21848064/21834768 [=====] - 0s 0us/step

```

[7] 1 from PIL import Image
2
3 img = Image.open("icecream.jpeg")
4 img.load()
5 data = np.asarray(img)
6 data = np.expand_dims(data, axis=0)
7 data.shape

```

(1, 224, 224, 3)

```

[17] 1 pred = model.predict(data)
2 max_args = (-pred).argsort()
3 max_args[0][:10]

```

array([928, 969, 927, 923, 961, 551, 960, 968, 438, 966])

شکل 7 - اجرای مدل

10 شماره کلاسی که بیشترین احتمال را داشتند در تصویر آمده است. این اعداد به ترتیب معادل icecream و trifle و plate و dough و egg nog هستند. همانطور که دیده می شود گزینه اول با بیشترین احتمال همان بستنی که در تصویر بوده است. موارد بعدی نیز تصاویر شبیه به آن دارند اما به درستی بستنی اولین گزینه است.

د) یک دسته از عکس های مختلفی که توسط مدل شما قابل تشخیص باشند، را به دلخواه انتخاب نمایید، پیش پردازش های لازم را انجام دهید و مدل را دوباره آموزش دهید. (من توانید از دیتاست های معروفی همچون **imageNet** استفاده نمایید و یا دیتاست های دیگری که به نظر مناسب من آیند)

دیتاست CIFAR-10 را در این بخش انتخاب می کنیم.

کلاس های موجود در این دیتاست شامل موارد زیر است.

جدول 1 - کلاس های موجود در دیتاست

dog	5	airplane	0
frog	6	automobile	1
horse	7	bird	2
ship	8	cat	3
truck	9	deer	4

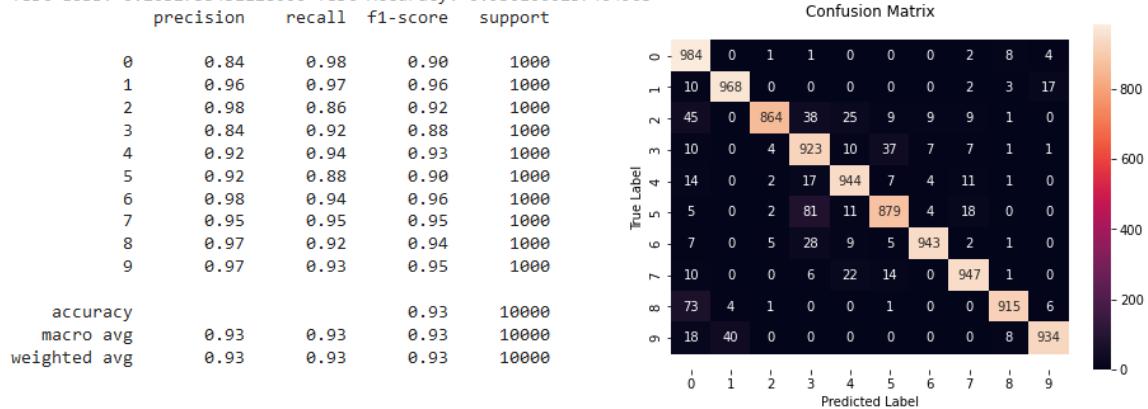
چون در بخش های قبلی هم از همین داده استفاده کردیم می توانیم به خوبی مقایسه کنیم که transfer learning چقدر می تواند مفید باشد. به علت محدودیت های سخت افزاری، مجبوریم فقط یک اپیاک آموزش را اجرا کنیم که البته چون داریم transfer learning انجام می دهیم کافی است و صرفا تغییرات کوچک باید انجام پذیرند.

نتیجه اجرا به شکل زیر است.

- خطأ: 0.2053
- دقت: 0.9301
- زمان کل آموزش: 451.15s در 1 اپیاک

313/313 [=====] - 20s 63ms/step - loss: 0.2053 - accuracy: 0.9301

Test Loss: 0.2052733451128006 Test Accuracy: 0.9301000237464905



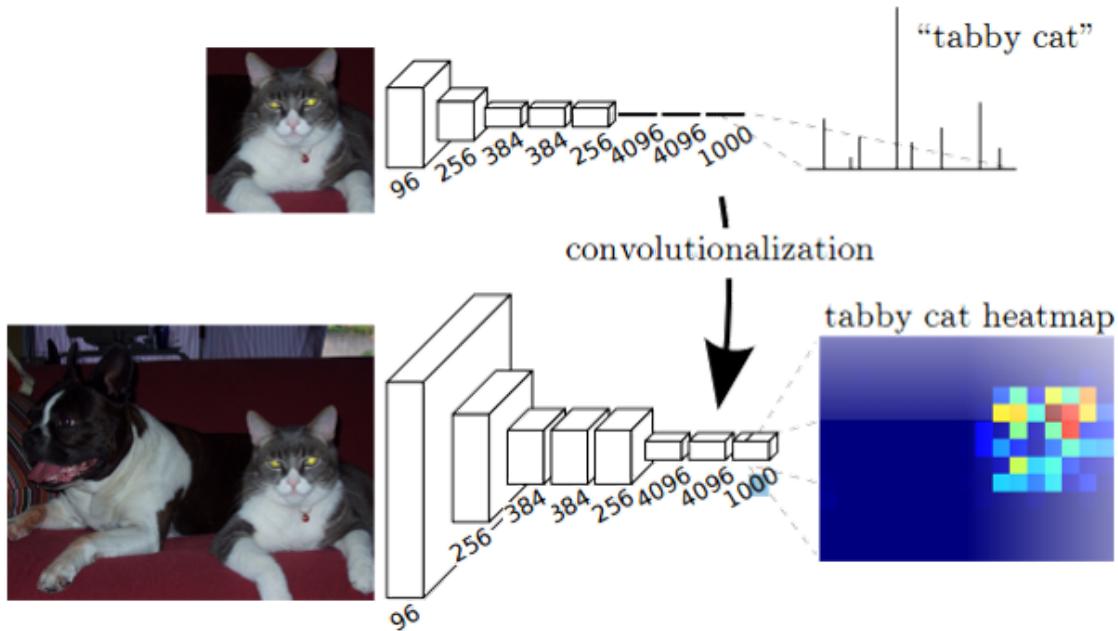
شکل 8 - نتیجه استفاده از transfer learning

همانطور که دیده می شود خطای دقت با اختلاف زیادی بهتر از CNN بخش های قبل که خودمان آموزش دادیم شده است. یک علت این است که مدل ما نسبتاً کوچک بود و این مدل 4 میلیون پارامتر دارد که خیلی بیشتر از مدل قبلی ما است. ضمناً دلیل دوم همین است که از transfer learning استفاده کردیم و وزنهای مدل آموزش دیده روی imagenet را بارگذاری کردیم. این کار باعث شده تا از دانش موجود در آن دیتابست بزرگ هم بهره ببریم که قطعاً خیلی روی نتیجه نهایی تاثیرگذار است.

سوال ۳ – Segmentation

در این سوال دو مدل FCN و DeepLabV3 را بررسی کرده و عملکرد این دو روش را بر روی یک تصویر واحد میسنجیم و با یکدیگر مقایسه می کنیم.

FCN .۱



شکل ۹ - ساختار FCN

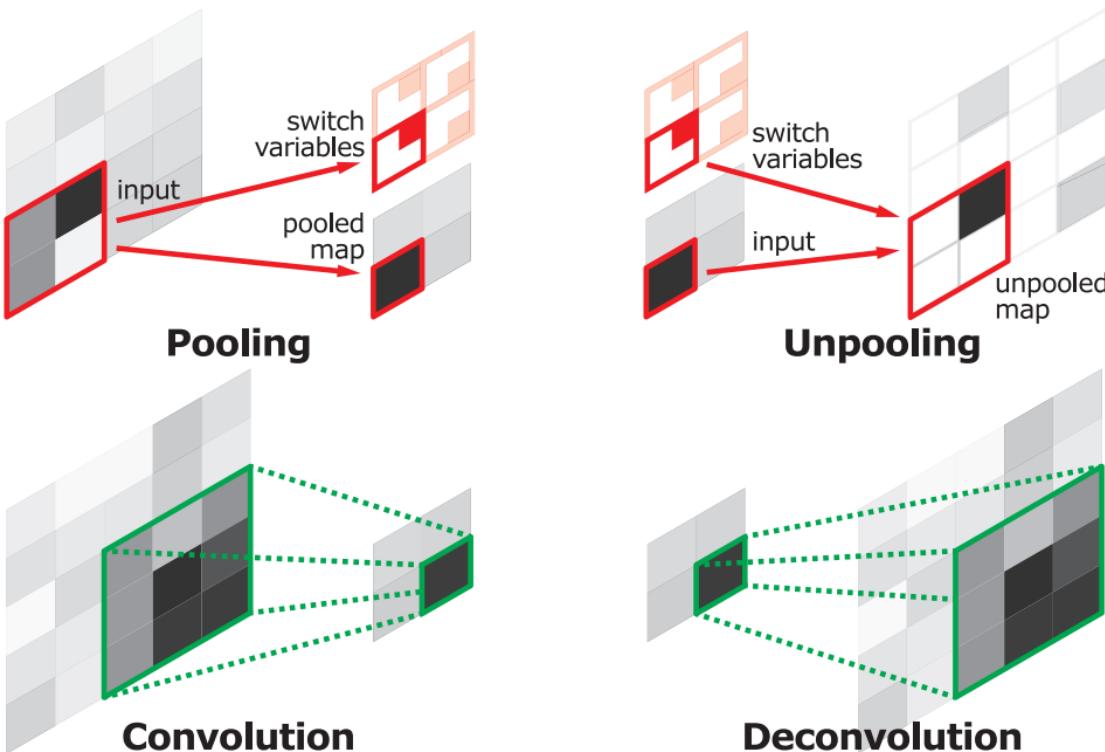
در روش های FCN یا Fully Convolutional Network عمدتا از سه روش Convolution، Upsample، Skip Layer استفاده می شود.

روش Convolution به این معنا است که از روش های fully connected قدیمی استفاده نکرده و به جای آن از لایه های convolutional استفاده کنیم.

Upsampling به معنای Deconvolution است. البته در فرمورک های مختلف با نام های متفاوتی یاد می شود. در Caffe و Kera به آن Deconvolution می گویند، در حالی که در tensorflow به آن conv_transpose گفته می شود.

همانطور که می دانیم، pooling اندازه تصویر را کاهش می دهد، برای مثال در VGG16 پس از پنج بار انجام pooling، اندازه تصویر ۳۲ برابر کاهش می یابد. برای به دست آوردن یک تصویر که عملیات segmentation روی آن انجام شده است، اندازه تصویر خروجی باید به بزرگی تصویر اصلی باشد. بنابراین باید upsampling یا همان deconvolution را انجام دهیم.

deconvolution مشابه کانولوشن است، با این تفاوت که اما کانولوشن چند به یک است و backward propagation یک به چند است. برای بخش های forward و backward propagation در دکانولوشن، معکوس همان forward and backward propagation ای که در کانولوشن استفاده می شود می تواند مورد استفاده قرار گیرد. بنابراین الگوریتم های optimization یا backward propagation به راحتی و بدون مشکل قابل انجام هستند. عملیات pooling به این منظور انجام می شود که موقعیت مقدار خروجی را هنگام pooling به خاطر بسپاریم و سپس این مقدار را بازیابی کنیم. تصویر زیر خلاصه ای از این عملیات است:

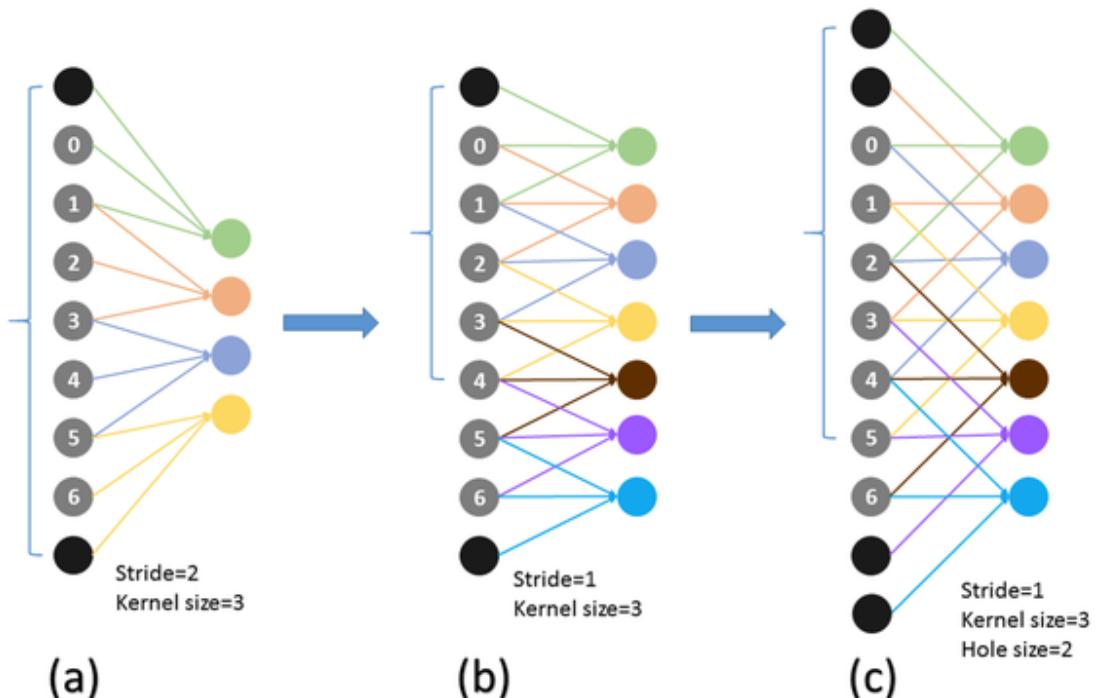


شکل 10 - روش *Deconvolution* و *Unpooling*

DeepLab .۲

DeepLab مدلی است که توسط گوگل پیشنهاد شده است. این مدل همچنان از FCN برای گرفتن score map ها استفاده می کند و همچنین شبکه VGG را fine-tuning می کند. اما روشی که با آن score map را پردازش می کند، بسیار ظریف تر و دقیق تر از روش های پردازشی FCN اصلی است.

روش Deeplab از یک رویکرد بسیار ظرف و دقیق استفاده می کند: stride یا همان گام لایه های pool4 و pool5 شبکه VGG از تعداد اصلی ۲ به عدد ۱ تغییر کرده است، و در عوض یک لایه padding اضافه شده است. این تغییر مانند این است که stride کل شبکه VGG از مقدار اصلی که ۳۲ بوده به ۸ تغییر می کند، به طوری که وقتی تصویر ورودی 514×514 است، یک score map با اندازه 67×67 خواهیم داشت که بسیار متراکم تر از FCN است.



شکل 11 - تغییرات جدید در لایه های DeepLab

حال که معناری هر دو مدل انتخابی خود را بررسی کرده و تفاوت آن ها را ذکر کردیم، اقدام به اجرای مدل ها بر روی یک تصویر یکسان می کنیم. تصویر انتخابی همان تصویر اولی است که در صورت سوال قرار دارد:



شکل 12 - تصویر ورودی برای مقایسه مدل ها

قبل از اجرای مدل های ذکر شده سعی می کنیم تا خروجی مورد نظر توسط هر مدل را پیشビینی کنیم.

همانطور که توضیح داده شد، مدل DeepLab از روش های دقیق تر و ظرفی تری برای پردازش score map استفاده می کند، به همین دلیل انتظار داریم که اشیا و مرز بین آن ها را دقیق تر و بهتر از هم جدا کند و در کل نیز خطای کمتری داشته باشد.

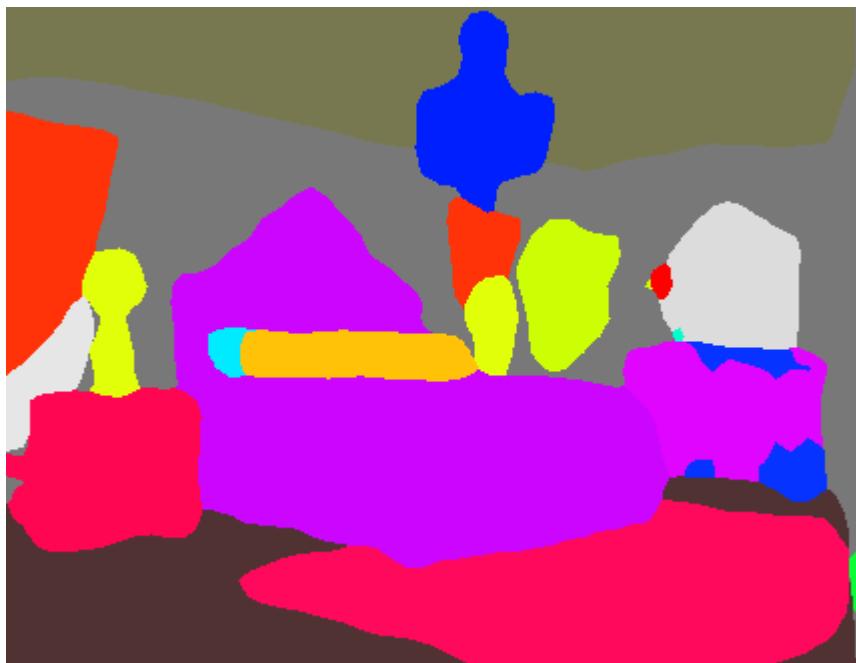
حال با استفاده از پکیج gluoncv در پایتون، مدل های از پیش تمرین داده شده را لود کرده و تصاویر را به عنوان ورودی به مدل می دهیم تا نتیجه segmentation را مشاهده کنیم:

نتیجه برای مدل FCN بصورت زیر است:



شکل 13 - نتیجه مدل *FCN*

همچنین تصویر خروجی مدل DeepLabv3 نیز به صورت زیر است:



شکل 14 - نتیجه مدل *DeepLabv3*

همانطور که حدس زده بودیم، روش DeepLab عملکرد بهتری از خود نشان داد و جزئیات اشیا را با دقت بیشتری از هم جدا کرد. برای مثال چراغ شب خواب کار پنجره در روش DeepLab دقیق تر از روش FCN تشخیص داده شده است.

سوال 4 - Object Detection

(الف)

۱. با اضافه کردن نرمال سازی دسته ای (batch normalization) تغییر در مقدار واحد در لایه پنهان کاهش می یابد و انجام این کار باعث بهبود پایداری شبکه عصبی می شود. با این کار بیش از ۲٪ بهبود در mAP (mean Average Precision) حاصل شد.

۲. این کار همچنین به منظم شدن مدل کمک کرد و میزان overfitting بطور کلی کاهش یافت. در YOLO High Resolution Classifier از ورودی با سایز 224×224 برای آموزش داده ها استفاده می شد، اما در YOLOv2 از ورودی های با اندازه 448×448 استفاده می شود. این تغییرات برای تنظیم دقیق شبکه طبقه بندی برای epoch 10 در دیتاست ImageNet بکارگیری شد.

۳. در YOLOv2 Convolutions with Anchor Boxes تمام لایه های YOLOv2 حذف شده اند و از Anchor Box ها برای پیش بینی bounding box ها استفاده می شود. همچنین یک لایه pooling برای افزایش رزولوشن خروجی حذف می شود.

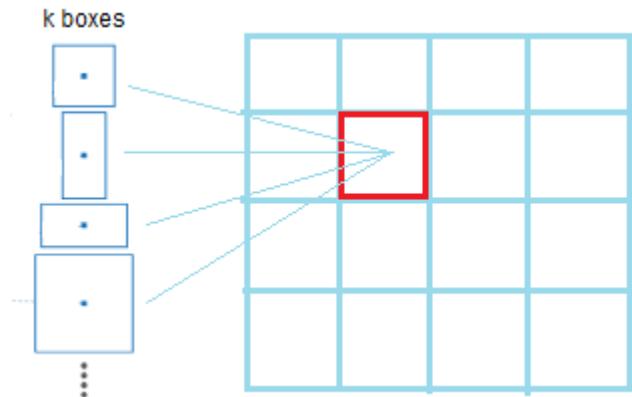


شکل 15 - نمونه ای از مشکل اختصاص هر سلول به یک شیء

روش YOLOv1 تلاش می کند که یک شی را به به آن grid cell ای اختصاص دهد که نقطه میانی جسم را در بر میگیرد. با استفاده از این ایده سلول قرمز در تصویر بالا باید هر دو شی مرد و کراوات را شناسایی کند، اما از آنجایی که هر سلول شبکه تنها می تواند یک شی را شناسایی کند، در اینجا با مشکل مواجه می شویم. برای حل این مشکل، روش YOLOv2 سعی میکند

تا به grid cell ها اجازه دهد که بتوانند بیش از یک شی را با استفاده از K تا bounding box را شناسایی کنند.

با استفاده از Anchor Box ها، مقدار mAP به میزان خیلی اندک کاهش یافت اما مقدار recall افزایش قابل توجهی پیدا کرد (از ۸۱ به ۸۸ درصد رسید).



شکل ۱۶ - بکارگیری Anchor Box در YOLOv2

۴. Fine-Grained Features: یکی از مسائل اصلی که در YOLO v1 وجود دارد، تشخیص اشیاء کوچکتر روی تصویر است. این مشکل در YOLO v2 حل شده است. به این صورت که تصویر را به grid cell های 13×13 تقسیم می کند که در مقایسه با نسخه قبلی آن کوچکتر است. این کار YOLOv2 را قادر می سازد تا اشیاء کوچکتر را در تصویر شناسایی یا مکان یابی کند و همچنین برای اشیاء بزرگتر نیز بطور مؤثر پاسخگو باشد. با این کار مقدار mAP به اندازه ۱ درصد افزایش یافت.

۵. Multi-Scale Training: روش YOLOv1 در تشخیص اشیاء با اندازه های ورودی مختلف دارای ضعف است. به این صورت که اگر برای آموزش یک شی خاص از تصویری با اندازه ی کوچک استفاده شود، در تشخیص همان شی در تصویر با اندازه ای بزرگتر به مشکل پر میخورد. این مشکل تا حد زیادی در YOLOv2 حل شده است. روش حل آن نیز به این صورت است که در YOLOv2، تصاویر تصادفی با ابعاد مختلف بین 320×320 تا 608×608 آموزش داده می شود.

۶. max pooling: YOLOv2 از معماری DarkNet 19 با ۱۹ لایه ی کانولوشن و ۵ لایه ی Softmax برای اشیائی که قرار است طبقه بندی شوند استفاده می کند. معماری Darknet 19 در زیر نشان داده شده است. Darknet 19 یک چارچوب شبکه عصبی است که در

زبان C و Cuda نوشته شده است. این معماری در تشخیص شی که برای پیش بینی در زمان کوتاه و real time اهمیت دارد، بسیار سریع است.

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

شکل 17 - معماری Darknet 19

در مجموع روش YOLOv2 بهتر، سریع تر و قوی تر از نسخه قبلی عمل می‌کند. به طوری که قادر به تشخیص و طبقه بندی اشیا با کانفیگ ها و ابعاد مختلف است. همچنین این روش در تشخیص اشیا کوچکتر بهبودی زیادی داشته و دقت آن در تشخیص اینگونه اشیا بیشتر شده است.

	YOLO								YOLOv2
batch norm?	✓	✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?		✓	✓	✓	✓	✓	✓	✓	✓
convolutional?		✓	✓	✓	✓	✓	✓	✓	✓
anchor boxes?		✓	✓						
new network?			✓	✓	✓	✓	✓	✓	✓
dimension priors?				✓	✓	✓	✓	✓	✓
location prediction?					✓	✓	✓	✓	✓
passthrough?						✓	✓	✓	✓
multi-scale?							✓	✓	✓
hi-res detector?								✓	
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	78.6

شکل 18 - بهبود مرحله به مرحله نسبت به YOLOv1 با استفاده از روش های جدید بکار رفته در YOLOv2

(ب)

	YOLO v5	Faster RCNN
Inference Speed	✓	
Detection of small or far away objects	✓	
Little to no overlapping boxes	✓	
Missed Objects	✗	✗

شکل 19 - مقایسه ویژگی های YOLOv5 با روش Faster CNN

تفاوت اصلی بین Yolov3، Yolov4 و معماری Yolov5 این است که Yolov3 از معماری CSPDARKNE53 در بدنه خود استفاده می کند. اما معماری YOLOV4 از DarkNet53 به عنوان بدنه اصلی خود استفاده می کند و Yolov5 نیز از ساختار Focus با معماری cspdarknet53 به عنوان ساختار اصلی استفاده می کند. لایه Focus برای اولین بار در Yolov5 معرفی شده است. لایه Focus جایگزین سه لایه اول در الگوریتم Yolov3 می باشد. مزیت استفاده از این معماری این می باشد که به حافظه CUDA کمتری نیاز داریم، همچنین لایه ها کاهش یافته و forward propagation و back propagation نیز بهبود می یابد.

ادعا می کند که بسیار سریع است، اندازه مدل بسیار سبکی دارد و از نظر دقت نیز با مدل YOLOv4 برابر می کند.

Metric	Scaled Yolov4 (Epochs=100)	Yolov5 (Epochs=50)
Precision	0.959	0.894
Recall	0.964	0.945
mAP@.5	0.981	0.954
mAP@.5:.95	0.791	0.762

شکل 20 - مقایسه دقت YOLOv5 و YOLOv4

	Scaled Yolov4 (Epochs=100)	Yolo v5 (Epochs=50)
Time to train	34min 13s	10min 17s
Time to run inference	17 - 24 ms	14 - 26 ms

شکل 21 - مقایسه سرعت YOLOv5 و YOLOv4

(پ)

در کل برای انجام تشخیص اشیا باید دو مسئله را حل کرده و انجام دهیم:

۱. اشیا مختلف موجود در تصویر را پیدا کنیم؛
 ۲. هر یک از آن اشیا را طبقه بندی کرده و اندازه آنها را با یک bounding box تخمین بزنیم.
- در مدل های تشخیص اشیایی دو مرحله ای مانند Fast R-CNN (شبکه های عصبی کانولوشن Mask R-CNN) یا Region-based RPN (Region-based Proposal Network) برای تولید مناطق مختلف و اشیا موجود استفاده می شود و سپس نتایج RPN برای طبقه بندی اشیا و یافتن bounding box های موجود استفاده می شود. چنین مدل هایی بالاترین میزان دقت را در تشخیص اشیا دارند، اما به طور معمول کندتر عمل می کنند.

مدل های تشخیص اشیا تک مرحله ای مانند YOLO و SSD (Single Shot MultiBox)، دو مرحله‌ی ذکر شده را در یک قدم انجام می‌دهند و در واقع عملیات تشخیص اشیا را مانند یک مسئله regression حل می‌کنند. به طوری که ورودی را دریافت کرده و شروع به یادگیری احتمال کلاس‌های مختلف و مختصات bounding box‌ها از بین مکان‌های ممکن می‌کنند. چنین مدل‌هایی به میزان دقیق پایین تری می‌رسند اما بسیار سریعتر از شبکه‌های تشخیص شیء دو مرحله‌ای هستند.

(ت)

۱. نصب پیش نیاز‌ها و بارگیری مجموعه دادگان

برای اینکه بتوانیم از مجموعه دادگان داده شده برای آموزش و تست در محیط colab استفاده کنیم، ابتدا نیاز داریم مجموعه دادگان را در وبسایت roboflow بازگذاری کرده و پیش پردازش‌های لازم را برای دادگان خود انجام دهیم.

در این سایت یک پروژه جدید ایجاد کرده و پوشه‌ی دیتاست داده شده را بطور کامل در آن بازگذاری می‌کنیم. در فایل data.yml، آدرس پوشه‌های داده‌های آموزش و ارزیابی داده شده است و این داده‌ها از قبل جدا شده‌اند.

داده‌ها در دو دسته‌ی آموزش و ارزیابی بازگذاری و توسط roboflow پیش پردازش می‌شوند و آماده‌ی استفاده در فرایند آموزش خواهند بود.

با استفاده از بخش Generate New Version، یک نسخه قابل استفاده از دیتاست خود را می‌سازیم که در واقع با استفاده از API Key ای که تولید می‌شود، می‌توان پروژه را در محیط پایتون لود کرده و سپس دیتاست را بازگیری کرد.

کد پیاده سازی الگوریتم YOLOv5 در آدرس <https://github.com/ultralytics/yolov5> قرار دارد. بنابراین ابتدا کدها را با استفاده از git دریافت کرده و از نسخه‌ای از کد که قصد داریم با آن کار کنیم، استفاده می‌کنیم.

در قدم بعدی نیازمندی‌های این پروژه را بر اساس فایل requirements.txt نصب می‌کنیم تا پروژه آماده‌ی استفاده شود.

۲. تعریف معماری و پیکربندی مدل

برای ساختن مدل مد نظر خود، به یک اسکریپت yaml نیاز داریم که پارامترهای مورد نیاز برای ساخت شبکه را مشخص کند. پارامترهایی مانند تعداد کلاس‌ها، anchor‌ها و مشخصات هر لایه.

برای این کار از فایل YOLOv5 کد yolov5/models/yolov5s.yaml موجود در سورس کد استفاده می‌کنیم. البته باید تعداد کلاس‌ها را تغییر دهیم و آن را بر اساس دیتابست خود تنظیم کنیم. تعداد کلاس‌ها در اینجا ۶ تا است که می‌توانیم آن را از فایل data.yml موجود در دیتابست نیز بخوانیم که در متغیر nc ذخیره شده است.

فایل yolov5s.yaml حاوی مشخصات شبکه است. در این فایل ابتدا تعداد کلاس‌ها مشخص شده و پس از آن مشخصات anchor‌ها ذکر شده است.

سپس لایه‌های backbone شبکه به ترتیب تعریف شده اند که از ماثوله‌های Focus، Conv، SPP و BottleneckCSP در لایه‌های مختلف استفاده شده است.

در نهایت نیز head شبکه YOLOv5 تعریف می‌شود تا با استفاده از آن بتوانیم تصاویر موجود در دیتابست را بعنوان ورودی به شبکه داده و آموزش خود را انجام دهیم.

۳. آموزش دادگان

برای انجام و شروع فرایند آموزش، ابتدا باید پارامترهای آموزش را مشخص کنیم.

پارامتر img مشخص کننده اندازه تصاویر دیتابست است که به عنوان ورودی به شبکه‌ی خود می‌دهیم. این مقدار برای دیتابست ما عدد ۴۱۶ است.

پارامتر batch size مقدار batch size را مشخص می‌کند که ما در اینجا از اندازه ۱۶ استفاده می‌کنیم.

سپس با استفاده از پارامتر epochs، تعداد دورهای یادگیری را ۱۰۰ دور می‌گذاریم. با پارامتر data نیز آدرس فایل data.yml که مشخص کننده آدرس دادگان آموزش و ارزیابی و نیز تعداد و نام کلاس‌ها بود را مشخص می‌کنیم.

با پارامتر cfg نیز آدرس فایل yaml که شامل پیکربندی شبکه بود را به عنوان ورودی می دهیم تا مورد استفاده قرار گیرد.

نام خروجی نتایج خود را با پارامتر name yolov5s_results انتخاب کرده و می گذاریم.

همچنین از پارامتر cache استفاده می کنیم تا تصاویر ذخیره و کش شوند و آموزش را سریعتر انجام دهیم.

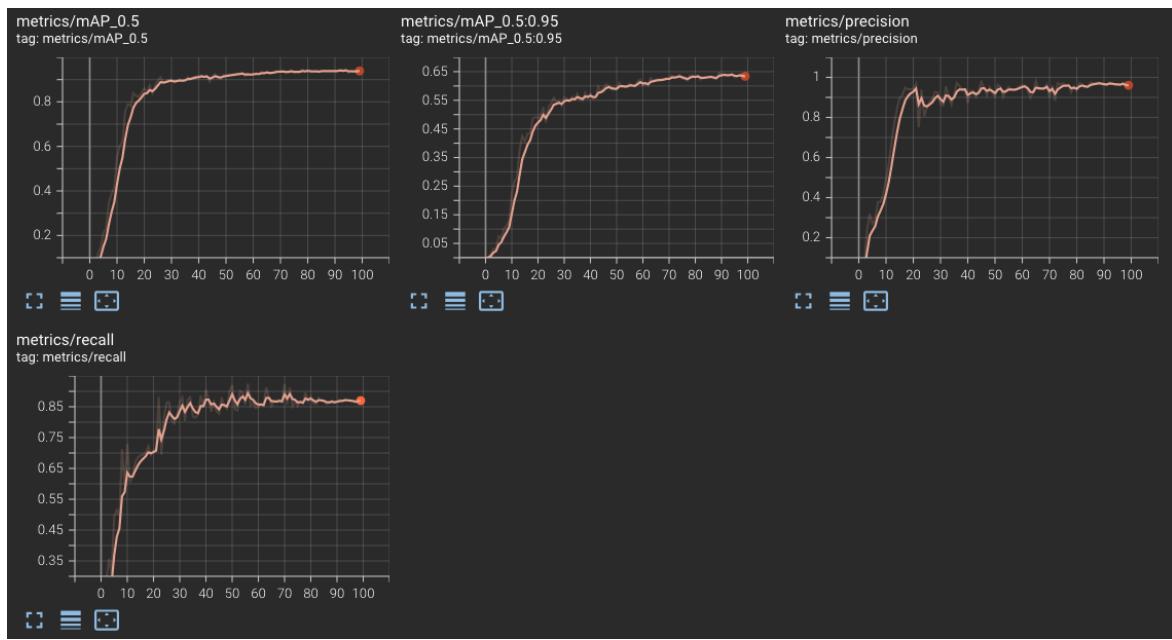
حال مدل خود را در ۱۰۰ دوره آموزش می دهیم و پس از ۱۰۲ ساعت، آموزش کامل می شود:

Epoch	gpu_mem	box	obj	cls	total	targets	img_size
96/99	1.81G	0.0434	0.03636	0.00715	0.08691	238	416: 100% 99/99 [00:41<00:00, 2.37it/s]
	Class	Images	Targets	P	R	mAP@.5	mAP@.5:.95: 100% 2/2 [00:01<00:00, 1.75it/s]
	all	58	1.00e+03	0.962	0.868	0.938	0.633
Epoch	gpu_mem	box	obj	cls	total	targets	img_size
97/99	1.82G	0.04438	0.03685	0.007484	0.08872	195	416: 100% 99/99 [00:41<00:00, 2.39it/s]
	Class	Images	Targets	P	R	mAP@.5	mAP@.5:.95: 100% 2/2 [00:01<00:00, 1.76it/s]
	all	58	1.00e+03	0.974	0.863	0.935	0.64
Epoch	gpu_mem	box	obj	cls	total	targets	img_size
98/99	1.82G	0.04345	0.03515	0.007145	0.08575	208	416: 100% 99/99 [00:41<00:00, 2.39it/s]
	Class	Images	Targets	P	R	mAP@.5	mAP@.5:.95: 100% 2/2 [00:01<00:00, 1.73it/s]
	all	58	1.00e+03	0.955	0.868	0.938	0.633
Epoch	gpu_mem	box	obj	cls	total	targets	img_size
99/99	1.82G	0.0443	0.03586	0.00738	0.08754	213	416: 100% 99/99 [00:41<00:00, 2.40it/s]
	Class	Images	Targets	P	R	mAP@.5	mAP@.5:.95: 100% 2/2 [00:02<00:00, 1.13s/it]
	all	58	1.00e+03	0.957	0.874	0.942	0.632
	blue	58	115	0.983	0.983	0.995	0.662
	green	58	290	0.996	0.983	0.992	0.659
	red	58	290	0.99	1	0.996	0.702
	vline	58	136	0.932	0.574	0.94	0.761
	white	58	58	0.853	0.702	0.731	0.277
	yellow	58	116	0.987	1	0.995	0.728
Optimizer stripped from runs/train/yolov5s_results/weights/last.pt, 14.8MB							
Optimizer stripped from runs/train/yolov5s_results/weights/best.pt, 14.8MB							
100 epochs completed in 1.218 hours.							
CPU times: user 57.7 s, sys: 8.37 s, total: 1min 6s							
Wall time: 1h 13min 51s							

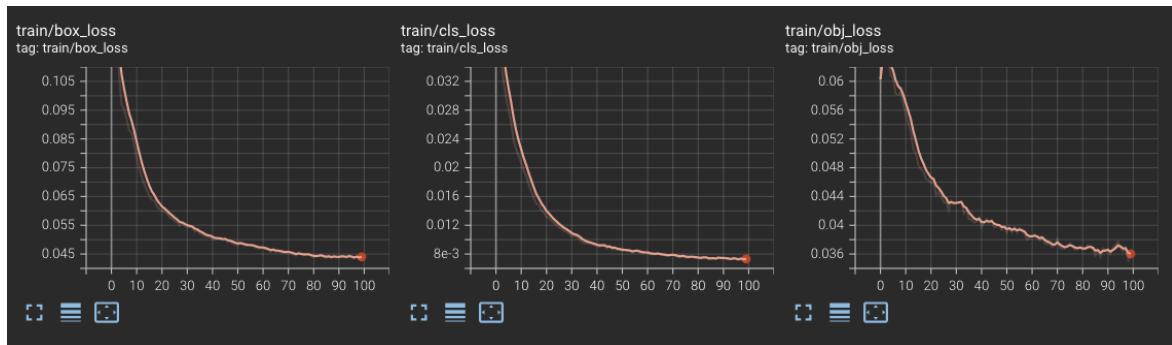
شکل 22 - جزئیات epoch ها در epoch های پایانی به همراه مدت زمان آموزش

۴. ارزیابی مدل

اکنون نتایج خود را که در yolov5s_results ذخیره کرده بودیم نمایش می دهیم تا فرایند یادگیری و کاهش loss و افزایش دقیق را در مدل خود مشاهده کنیم:



شکل 23 - نمودارهای دقیقی YOLOv5 برای داده های آموزش

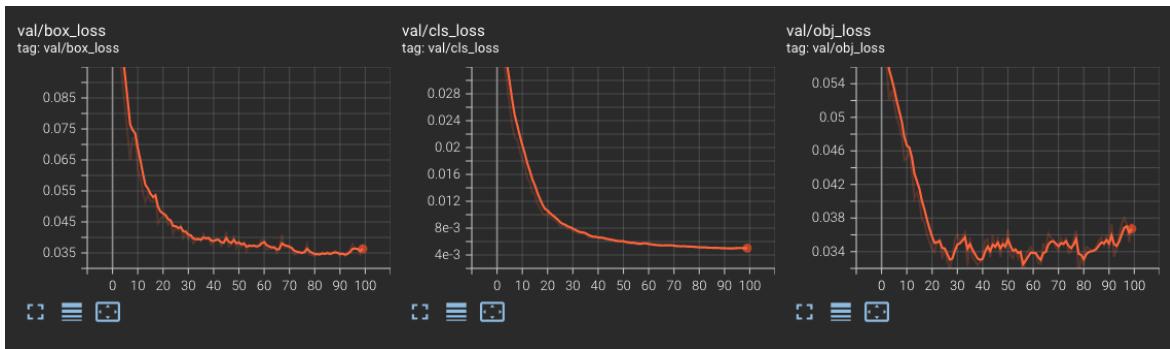


شکل 24 - نمودارهای خطا های YOLOv5 برای داده های آموزش

همانطور که مشاهده می کنیم مدل به خوبی آموزش دیده و مقدار loss برای آن در هر epoch رفته کاهش یافته و به صفر نزدیک شده است.

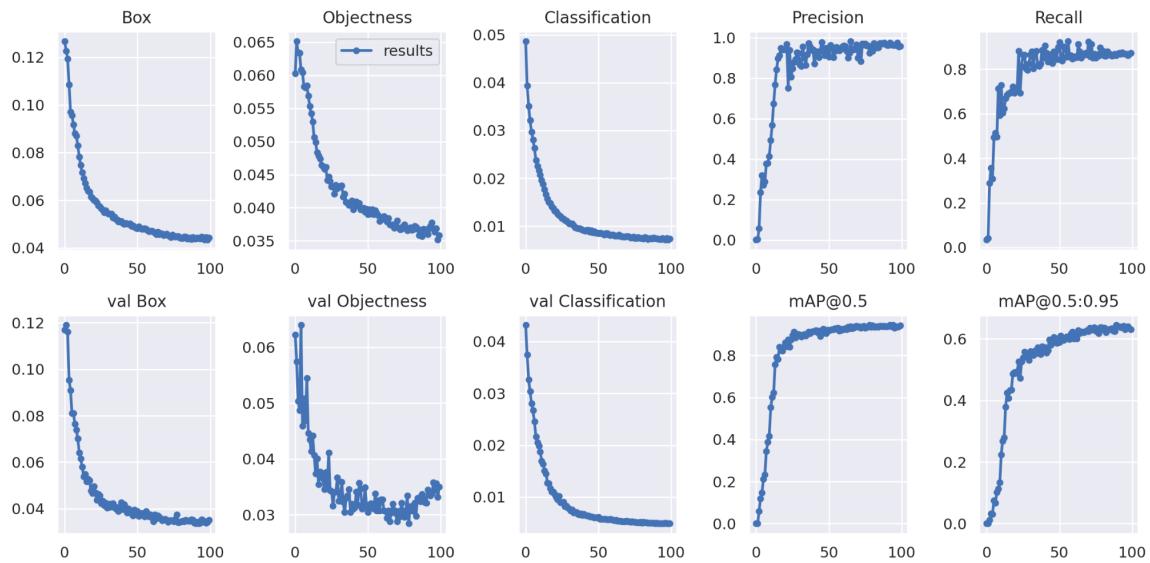
همچنین معیار های mAP و recall, precision نیز رفته بیشتر شده اند و به اعداد قابل قبولی رسیده اند.

حال نمودار تغییرات خطا را برای داده های ارزیابی مشاهده می کنیم:



شکل 25 - نمودارهای خطای خطا برای داده های ارزیابی

در زیر نیز تمامی نمودار های خطای داده های آموزش و ارزیابی در کنار هم مشاهده میکنیم:



شکل 26 - تجمعی نمودارهای دقت و خطای برای داده های آموزش و ارزیابی

همانطور که در نمودار های بالا مشاهده می شود، نمودارهای خطای داده های ارزیابی نیز به صورت کاهشی بوده و به صفر نزدیک شده است. البته در نمودار Objectness میبینیم که در epoch های پایانی مقدار خطای بیشتر شده و به نوعی overfitting در حال رخ دادن می باشد. نمودار های دقت نیز افزایشی بوده و precision به نزدیک ۱۰۰ درصد رسیده است. مقدار recall نیز حدود ۹۵ درصد است.

۵. نمایش تصاویر برچسب گذاری شده

در این قسمت تعدادی از تصاویر که اشیا مختلف در آن تشخیص داده شده اند را به همراه دسته ای اشیا نمایش می دهیم.

در دیتاست ما ۶ دسته وجود داشت که توب های بیلیارد با رنگ های متفاوت را شامل می شد.

نتایج برچسب گذاری برای تعدادی از داده های ارزیابی بصورت زیر است:



شکل 27 - بررسی خروجی تولید شده توسط مدل برای تعدادی از داده های ارزیابی

همانطور که مشاهده می کنیم عملیات تشخیص شی به خوبی انجام شده و هر توب در کلاس رنگ خود به درستی و با دقت بالایی قرار گرفته است.

حال این کار را برای داده های آموزشی که با استفاده از تکنیک augmentation تولید شده اند انجام می دهیم و نتیجه به صورت زیر خواهد شد:



شکل 28 - بررسی خروجی تولید شده توسط مدل برای تعدادی از داده های آموزش که با روش Augmentation تولید شده اند.

مشاهده می کنیم که حتی وقتی بیش از یک شی از همان جنس نیز در تصویر وجود دارد، آن شی به خوبی تشخیص داده شده و دسته ای آن نیز به درستی انتخاب شده است.

۶. ذخیره کردن مدل

در نهایت می توانیم مدل آموزش دیده ای خود را در یک فایل ذخیره کنیم و در آینده بدون نیاز به اجرای مجدد فرایند آموزش، از مدل استفاده کرده و عملیات تشخیص شی را انجام دهیم.

بازخورد

با سلام و وقت بخیر

این تمرین از نظر اینکه با مدل‌های بزرگ باید کار می‌کردیم و معمولاً این مدلها روی سیستم‌های عادی به کندی آموزش می‌بینند و یا حتی امکان بارگذاری ندارند، اگر مدلها و دیتاست‌ها دقیقاً مشخص می‌شدند تا نیاز نباشد ما امکان آموزش را بررسی کنیم بهتر می‌شد.

با تشکر از وقتی که گذاشتید