دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی برق

گزارش‌کار تمرین ۲

پیاده سازی CNNو MLP

نگارش

علی بابالو

استاد راهنما

سرکار خانم دکتر سیدین

دی ماه 1401

چکیده

در این آزمایش هدف پیاده سازی یک شبکه کانولوشنی روی(سی فار 10) است. شبکه عصبی کانولوشنی از پرکاربردترین شبکه های عصبی برای پردازش تصویر می باشد. عملکرد بسیاری از حملات نیز بر روی این شبکه عصبی بررسی می شوند .

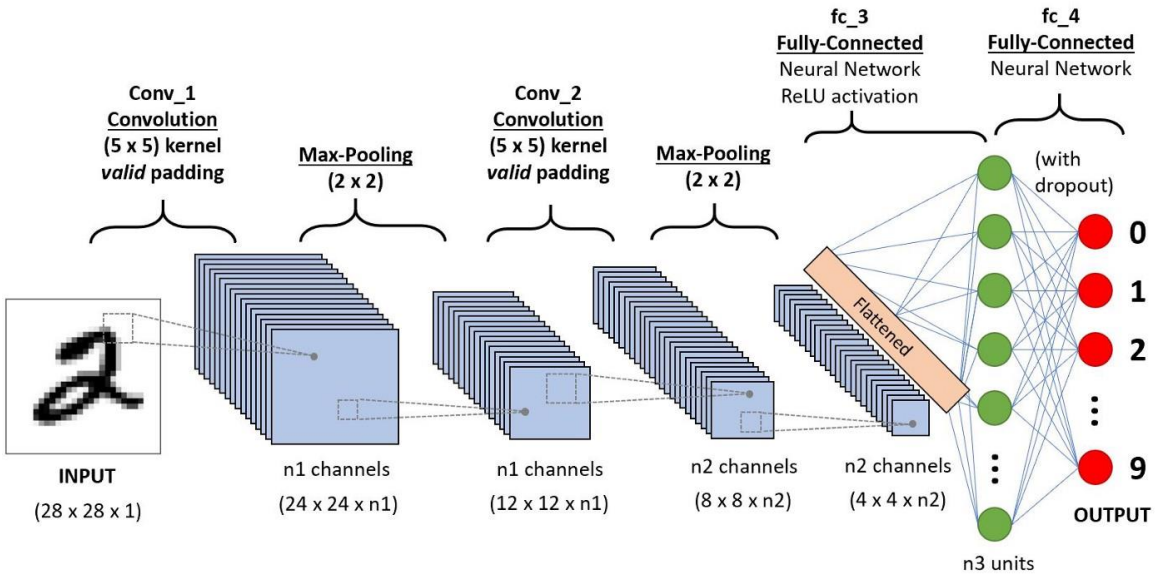
فهرست مطالب

صفحه

پیش گزارش

2-1- ساختار یک شبکه عصبی CNN.

شبکه کانولوشنی از لایه های اصلی زیر تشکیل شده است:

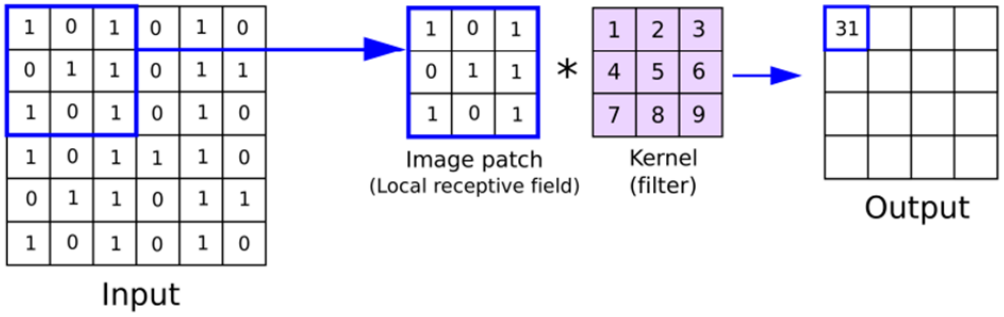


شکل 1) نمونه ای شبکه عصبی کانوولوشنی

- لایه کانولوشنی
- لایه ادغام
- لایه کاملا متصل

2-3-1- لایه کانولوشنی:

در این لایه عمل کانولوشن بر روی داده ورودی و با استفاده از تعدادی فیلتر انجام می شود. برای محاسبه هر درایه خروجی، ماتریس فیلتر بر روی ماتریس ورودی لغزانده می شود. عمل کانولوشن به این صورت تعریف می شود که ابتدا اولین عنصر فیلتر بر روی اولین عنصر ماتریس ورودی قرار می گیرد. سپس مجموع ضرب درایه های متناظر فیلتر با درایه های متناظز ماتریس ورودی محاسبه می شود. در نهایت فیلتر بر روی تصویر ورودی به اندازه پارامتر از پیش تعیین شده s به جلو برده می شود. با تکرار این مراحل ماتریس خروجی محاسبه می شود.



شکل 2) عملگر کانوولوشن

2-3-2- لایه ادغام:

یکی دیگر از لایه های شبکه کانولوشنی لایه ادغام است. این لایه هیچ پارامتر آموزشی ندارد. هدف این لایه کاهش ابعاد ماتریس ورودی ورودی و همزمان حفظ اطلاعات ارزشمند ورودی است. در این لایه ابتدا ماتریس با ابعاد از پیش تعیین شده k در نظر گرفته می شود. این پارامتر معمولا برابر با ۲ در نظر گرفته می شود. سپس با لغزاندن فیلتر بر روی ورودی اندازه ماتریس ورودی کاهش می یابد. یک نمونه از عملگرهایی که برای نمونه برداری در این لایه استفاده می شود، عملگر بیشینه است. در این حالت ماتریس از پیش تعیین شده بر روی داده ورودی لغزانده می شود و تنها بیشینه عناصری که در هر بخش قرار می گیرند را به عنوان خروجی در نظر می گیرد.

3-3-2- لایه کاملا متصل:

در این لایه یک شبکه عصبی کاملا متصل قرار گرفته است. در این لایه هدف مرتبط کردن ماتریس نهایی با خروجی های نهایی شبکه است. وزن های شبکه کاملا متصل از طریق پس انتشار خطا بدست می آید.

شرح آزمایش

CIFAR مخفف عبارت Canadian Institute for Advanced Research است و مجموعه داده CIFAR-10 همراه با مجموعه داده CIFAR-100 توسط محققان موسسه CIFAR توسعه یافته است.

مجموعه داده شامل 60000 عکس رنگی 32×32 پیکسل از اشیاء از 10 کلاس، مانند قورباغه، پرندگان، گربه‌ها، کشتی‌ها و غیره است. برچسب‌های کلاس و مقادیر عدد صحیح مرتبط با آنها در زیر فهرست شده‌اند.

1. هواپیما
2. خودرو
3. پرنده
4. گربه
5. آهو
6. سگ
7. قورباغه
8. اسب
9. کشتی
10. کامیون

این تصاویر بسیار کوچک هستند، بسیار کوچکتر از یک عکس معمولی، و مجموعه داده برای تحقیقات بینایی کامپیوتری در نظر گرفته شده است.

CIFAR-10 یک مجموعه داده کاملاً درک شده است که به طور گسترده برای محک زدن الگوریتم‌های بینایی رایانه در یادگیری ماشین استفاده می‌شود. مشکل حل شده است." دستیابی به دقت طبقه بندی 80 درصد نسبتاً ساده است. عملکرد برتر در مورد مشکل با یادگیری عمیق شبکه های عصبی کانولوشن با دقت طبقه بندی بالای 90٪ در مجموعه داده آزمایشی به دست می آید.

اکنون می‌توانیم یک مدل پایه برای مجموعه داده CIFAR-10 بررسی کنیم.

2-2- معماری VGG

یک مدل پایه حداقل عملکرد مدل را ایجاد می کند که همه مدل های دیگر ما را می توان با آن مقایسه کرد و یک معماری مدلی که می توانیم به عنوان مبنای مطالعه و بهبود استفاده کنیم.

یک نقطه شروع خوب، اصول کلی معماری مدل های VGG است. اینها نقاط شروع خوبی هستند زیرا در رقابت ILSVRC 2014 به عملکرد برتر دست یافتند و به دلیل اینکه ساختار مدولار معماری قابل درک و پیاده سازی آسان است.

این معماری شامل انباشته شدن لایه‌های کانولوشن با فیلترهای کوچک 3×3 و به دنبال آن یک لایه ترکیبی حداکثر است. این لایه‌ها با هم یک بلوک را تشکیل می‌دهند و این بلوک‌ها می‌توانند در جایی که تعداد فیلترها در هر بلوک با عمق شبکه افزایش می‌یابد، تکرار شوند، مانند 32، 64، 128 و 256 برای چهار بلوک اول مدل. بالشتک روی لایه های کانولوشن برای اطمینان از مطابقت ارتفاع و عرض نقشه های ویژگی خروجی با ورودی ها استفاده می شود.

ما این معماری را در مسئله CIFAR-10 بررسی می کنیم و مدلی را با این معماری با بلوک های 1، 2 و 3 مقایسه می کنیم.

2-3- Dropout regularization از جمله راه های بهبود مدل

Dropout یک تکنیک ساده است که به طور تصادفی گره ها را از شبکه خارج می کند. این یک اثر منظم کننده دارد زیرا گره های باقی مانده باید برای برداشتن سستی گره های حذف شده سازگار شوند.

Dropout را می توان با افزودن لایه های Dropout جدید به مدل اضـافه کرد، جایی که مقدار گره های حذف شـده به عنوان یک پارامتر مشـخص می شـود. الگوهای زیادی برای افزودن Dropout به یک مدل وجود دارد، از نظر اینکه در کجای مدل باید لایه‌ها را اضافه کرد و از چه تعداد حذفی استفاده کرد.

در این حالت، لایه‌های Dropout را بعد از هر لایه جمع‌آوری حداکثر و بعد از لایه کاملاً متصـل اضـافه می‌کنیم و از نرخ خروج ثابت 20٪ استفاده می‌کنیم (به عنوان مثال، 80٪ گره‌ها را حفظ می‌کنیم).

2-4- تنوع dropout regularization

یک تغییر این است که میزان dropout از 20٪ به 25٪ یا 30٪ افزایش پیدا کند. یکی دیگر از تغییراتی که ممکن است جالب باشد، استفاده از الگوی افزایش dropout از 20٪ برای بلوک اول، 30٪ برای بلوک دوم و به همین ترتیب به 50٪ در لایه کاملا متصل در بخش طبقه‌بندی‌کننده مدل است.

این نوع افزایش dropout با عمق مدل یک الگوی معمولی اسـت. موثر اسـت زیرا لایه‌های عمیق مدل را مجبور می‌کند تا بیش از لایه‌های نزدیک به ورودی را منظم کنند.

2-5- استفاده از Batch Normalization

نرمال سازی دسته ای در تلاش برای تثبیت یادگیری و شاید تسریع روند یادگیری اضافه شده است. برای جبران این شتاب، از الگوی dropout فزاینده استفاده می شود.

در ادامه به دو راهکار دیگر که برای تقویت این مدل استفاده شده است نیز پرداخته شده است؛

2-6- استفاده از Adam learning rate

بهینه‌سـازی آدام یک روش stochastic gradient descent اسـت که مبتنی بر تخمین تطبیقی ممان‌های مرتبه اول و مرتبه دوم اسـت. به خاطر ذات تطبیقی یا adaptiveبسـیار مورد اسـتفاده قرار می‌گیرد به گونه ای که به گفته کینگما و همکاران، 2014، این روش "از نظر محاسـباتی کارآمد اسـت، نیاز به حافظه کمی دارد، نسبت به مقیاس مجدد مورب گرادیان ها ثابت است، و برای مسائلی که از نظر داده/پارامترها بزرگ هستند، مناسب است."

2-1- محیط Python

به طور کلی مراحل زیر در پیاده سازی این پروژه طی شده است؛

2-1-1 دیتاست Hotel

توضیحات مربوط به این دیتاست در دستورکار گفته شده است پس از توضیح دادن آن پرهیز می‌کنیم.

- **Dataset Preparation**
  - Data Preprocessing

بعد از اضافه کردن ایمپورت کردن کتابخانه ها مورد نیاز در ابتدای کد، به قسمت خواندن دیتاست و انجام تعییرات به روی آن می‌پردازیم. ابتدا دیتاست های آموزش و تست را با read_csv به یک دیتافریم تبدیل میکنیم و دیتاهایی با مقدار NaN را با ()dropna حذف می‌کنیم. سپس

  - Shuffling The Dataset

سپس با دستور ()df.sample و با نسبت ۱ دیتا را شافل می‌کنیم.

  - Categorical And Numerical Features

در ابتدا نام تمامی فیچر ها و سپس فیچر های عددی و فیچر های کتگوریکال را چاپ می‌کنیم تا متوجه شویم کدام فیچر هارا باید به عدد تبدیل کنیم.

برای تبدیل کردن فیچر هایی که دیتاتایپ آنها object است از ۲ روش بهره بردیم، روش اول که بهینه نمی‌باشد بدین ترتیب است که تمام مقادیر غیریکسان فیچر را بصورت دستی مقدار دهیم که اینکار را فقط برای فیچر ArrivalDateMonth انجام دادیم اما همانطور که گفتیم برای تعداد فیچر های زیاد و تعداد سمپل های زیاد کار وقت گیری است و به صرفه نیست پس برای همین علت از روش دوم استفاده کردیم که در آن ابتدا فیچرهایی که مقادیر آبجکت دارند را به دیتا categorical تبدیل می‌کنیم تا از متد cat.codes که برای دیتاهای categorical تعریف شده استفاده می‌کنیم. این متد یک سری از کدها را برای دیتا های متفاوت در نظر می‌گیرد و به آنها نسبت می‌دهد که خروجی در زیر سلول نشان داده شده است.

برای دیتاست تست هم مانند دیتا آموزش رفتار می‌کنیم و بجز فیچر ArrivalDateMonth باقی فیچر های دارای آبجکت را با cat.codes به عدد تبدیل می‌کنیم. اما تفاوت اصلی دیتاست تست و آموزش در مقادیر آنها در فیچر های مختلف است به عنوان مثال تعداد کشور های ذکر شده در دیتای تست با تعداد کشور های دیتا آموزش متفاوت است و از آن بیشتر است. برای حل آن از یک تابع لامبدا استفاده می‌کنیم به این صورت که اگر دیتای فیچر در دیتاست ترین قرار داشت که مقدار همان را بهش می‌دهیم اگر وجود نداشت مقدار -۱ را برای آن قرار می‌دهیم و این تابع را با متد apply به دیتاست وارد می‌کنیم.

- Normalizing Dataset

برای نرمالایز کردن دیتای آموزش از تابع Preprocessing در کتابخانه Sklearn استفاده می‌کنیم که دیتاست را نرمالایز می‌کند

- Split Dataset to Train and Validation

برای جدا کردن دیتاست به بخش آموزش و ولیدیشن نیز از متد sample استفاده می‌کنیم با تسبت .۰.۸ به ۰.۲ و سپس به جداسازی لیبل خروجی از باقی دیتاست می‌پردازیم بدین صورت که فیجر ADR را به لیبل خروجی و باقی فیچر هارا به ورودی شبکه می‌دهیم.

- **Building The Multilayer Perceptron**

در این بخش به آموزش شبکه می‌پردازیم:

- Model Building

برای بیلد کردن مدل از Sequential_API استفاده می‌کنیم که در آن از ۲ لایه مخفی با تعداد ۲۰ نورون استفاده می‌کنیم که تالع فعال ساز های آنها نیز RELU می‌باشد.

- Compile The Model

برای کامپایل کردن مدل شبکه نیز آپتیمایزر آدام و متریکس های MAE, MSE استفاده می‌کنیم.

- Train The Model

در این قسمت شبکه را با دیتای ترین، آموزش می‌دهیم و برای اینکار از تعداد Epoch های ۳۰ و ۵۰ و ۱۰۰ با batch size = 64 استفاده می‌کنیم که اوردر خطای هر کدام بترتیب ۰.۰۰۲ و ۰.۰۰۱۸ و ۰.۰۰۱۵ می‌باشد.

- Plotting

در قسمت پلات هم مقادیر MAE, MSE را بر حسب اپوک ترسیم می‌کنیم.

- Model Evaluation

در این قسمت با متد Evaluation خطا دیتای ولیدیشن را بر روی شبکه محاسبه می‌کنیم که برابر با ۰.۰۰۱۴ می‌باشد.

- **Model Prediction**

در این قسمت با استفاده از تابع predict مقدار ADR هر دیتا را برحسب فیچرهایش بر روی دیتای تست محاسبه می کنیم. که مقدار پیش بینی شده و مقدار واقعی در این مدل شبکه ما تمامی فیچر ها در فایل result.csv ذخیره شده است.

- **Feature Selection**

در قسمت د تمرین اول گفته شده کدام فیچر ها ببشترین تاثیر در یادگیری شبکه را دارند و چگونه آنها را جداسازی کنیم. برای اینکار از ماتریس Correlation استفاده کردم. برای این منظور تمام فیچر هایی که مقدار همبستگی آن با لیبل خروجی یعنی ADR بیشتر از مقدار دلخواه بود را به عنوان فیچر با تاثیر زیادتر تعیین می‌کنم. در اینجا مقدار همبستگی بیشتر از ۰.۱ را انتخاب کردم که فیچر های انتخاب شده را مشاهده میکنید.

- Train With Selected Features

سپس شبکه را با دیتاست جدید که فقط شامل فیچر های انتخابی است آموزش می‌دهیم که خروجی آن را نیز مانند قسمت قبلی پیشبینی می‌کنیم و آن را در فایلresult.csv ذخیره می‌کنیم.

2-1-2 دیتاست CIFAR-10

از این دیتاست در گذشته در درس هوش محاسباتی خانم دکتر عبداللهی استفاده کرده بودیم که مدل شبکه طراحی شده را و گزارشش را در پایین به صورت کامل مشاهده می‌کنید. برای تمرین دوم نیز برای مشاهده کد وارد قسمت ML Second Exercise بشوید. در این قسمت صرفا کد را یک برسی کوتاه می‌کنیم و برای برسی کامل می‌توانید گزارش پایین را بخوانید.

بعد از خواندن دیتاتست و اسکیل کردن آن شبکه را طراحی، کامپایل می‌کنیم و برای آموزش شبکه از batch size های ۳۲و۶۴و۱۲۸ تایی با تابع فعال ساز رلو استفاده کردیم و در مرحله بعد نیز همان شبکه را با تابع فعال ساز های tanh, sigmoid تست کردیم که نتایج مشخص هستند سپس مدل را evaluate کردیم و آن را تست کردیم که درنهایت آنرا مشاهده می‌کنید

2-1-2 دیتاست CIFAR-10

توضیحات مربوط به دیتاست cifar-10 در پیش گزارش داده شده است پس مستقیم به توضیح کد می‌پردازیم.

در ابتدا مانند دیتاست mnist کتابخانه هارا ایمپورت می‌کنیم و دیتاست را ادد می‌کنیم.

CNN classifier for the cifar10 dataset ¶

Importing libraries

[1]:
```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Dataset preperation (Preprocessing)

[2]:
```
cifar10_data = tf.keras.datasets.cifar10
(x_train, y_train), (x_test, y_test) = cifar10_data.load_data()
assert x_train.shape == (50000, 32, 32, 3)
assert x_test.shape == (10000, 32, 32, 3)
assert y_train.shape == (50000, 1)
assert y_test.shape == (10000, 1)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 13s 0us/step
```

Dataset scailing

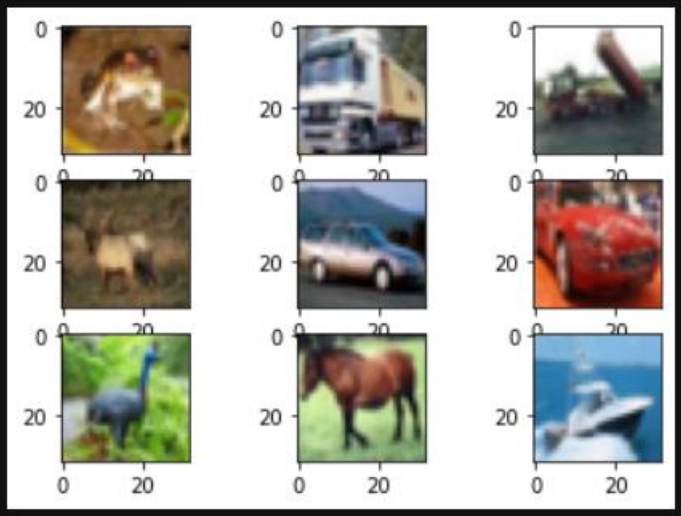
[3]:
```
def scale_data(train_images, test_images):
    """
    This function takes in the training and test images as loaded in the cell above, and scales them
    so that they have minimum and maximum values equal to 0 and 1 respectively.
    Your function should return a tuple (train_images, test_images) of scaled training and test images.
    """
    train_images = train_images/255.
    test_images = test_images/255.

    return (train_images, test_images)
```

[4]:
```
x_train_s, x_test_s = scale_data(x_train, x_test)
print(f"x_train_s shape:\n{x_train_s.shape}")
print(f"x_test_s shape:\n{x_test_s.shape}")
```

```
x_train_s shape:
(50000, 32, 32, 3)
x_test_s shape:
(10000, 32, 32, 3)
```

شکل۱۱) ایمپورت کردن دیتاست و کتابخانه ها

```
[6]: # plotting first few images
     for i in range(9):
      # define subplot
      plt.subplot(330 + 1 + i)
      # plot raw pixel data
      plt.imshow(x_train_s[i])
     # show the figure
     plt.show()
```



شکل۱۲) مشاهده چند نمونه از دیتاهای cifar-10

در مرحله بعدی به بیلد کردن شبکه مشغول می‌شویم. در اینجا نیز از API sequential استفاده می‌کنیم و لایه های مختلف را به شبکه اضافه می‌کنیم. برای اینکه تاثیر افزایش لایه ها را متوجه بشویم ۳ مدل استفاده می‌کنیم که هر کدام به اندازه شمارشان VGG دارند. در مدل۱ علاوه بر نرمالایزکردن بچ ها و لایه maxpooling از ۲ لایه conv2D استفاده کردیم که به تعداد ۳۲ فیلتر دارد و سایز آن ۳*۳ می‌باشد. در لایه مخفی اول نیز از ۱۲۸ نورون با تابع فعال ساز relu و لایه اخر نیز از تابع فعال سازsoftmax استفاده کردیم. در مدل ۲ علاوه بر VGG استفاده شد مدل ۱ از ۲لایه conv2D با ۶۴ فیلتر و تابع فعال ساز relu استفاده شده است و در مدل سوم نیز علاوه بر استفاده از مدل اول از ۲ لایه conv2D با ۱۲۸ فیلتر و با همان تابع فعال ساز relu استفاده کردیم که در ادامه تاثیر این افزایش لایه هارا مشاهده می‌کنیم.

Building the convolutional neural network models

Model_1 (1 VGG layer)

[7]:
```
def cnn_get_model_1(input_shape):
    """
    This function should build a Sequential model according to the above specification. Ensure the
    weights are initialised by providing the input_shape argument in the first layer, given by the
    function argument.
    Your function should return the model.
    """
    model = tf.keras.Sequential()

    # VGG_1
    model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=input_shape))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=input_shape))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
    model.add(tf.keras.layers.Dropout(0.2))

    # Output
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Dropout(0.5))
    model.add(tf.keras.layers.Dense(10, activation='softmax'))
    return model
```

شکل۱۳) تابع برای مدل اول شبکه

Model_2 (2 VGG layers)

[8]:
```
def cnn_get_model_2(input_shape):
    """
    This function should build a Sequential model according to the above specification. Ensure the
    weights are initialised by providing the input_shape argument in the first layer, given by the
    function argument.
    Your function should return the model.
    """
    model = tf.keras.Sequential()

    # VGG_1
    model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=input_shape))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=input_shape))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
    model.add(tf.keras.layers.Dropout(0.2))

    # VGG_2
    model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=input_shape))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=input_shape))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
    model.add(tf.keras.layers.Dropout(0.3))

    # Output
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Dropout(0.5))
    model.add(tf.keras.layers.Dense(10, activation='softmax'))
    return model
```

شکل۱۴) تابع برای بیلد کردن شبکه دوم

Model_3 (3 VGG layers)

```
[9]: def cnn_get_model_3(input_shape):
    """
    This function should build a Sequential model according to the above specification. Ensure the
    weights are initialised by providing the input_shape argument in the first layer, given by the
    function argument.
    Your function should return the model.
    """
    model = tf.keras.Sequential()

    # VGG_1
    model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=input_shape))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=input_shape))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
    model.add(tf.keras.layers.Dropout(0.2))

    # VGG_2
    model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=input_shape))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=input_shape))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
    model.add(tf.keras.layers.Dropout(0.3))

    # VGG_3
    model.add(tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=input_shape))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=input_shape))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
    model.add(tf.keras.layers.Dropout(0.4))

    # Output
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Dropout(0.5))
    model.add(tf.keras.layers.Dense(10, activation='softmax'))
    return model
```

شکل۱۵)تابع برای بیلد کردن شبکه سوم

```
[10]: model_1 = cnn_get_model_1(x_train_s[0].shape)
      model_2 = cnn_get_model_2(x_train_s[0].shape)
      model_3 = cnn_get_model_3(x_train_s[0].shape)
```

شکل۱۶) ۳ مدل را با استفاده از توابعشان تعریف می‌کنیم.

در ادامه نیز می‌توانید سامری شبکه هارا در کد مشاهده کنید که در شکل پایین لایه های شبکه مدل ۲ را مشاهده می‌کنید.

```
In [12]: model_2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 32, 32, 32)        896

batch_normalization_3 (Batc  (None, 32, 32, 32)        128
hNormalization)

conv2d_3 (Conv2D)            (None, 32, 32, 32)        9248

batch_normalization_4 (Batc  (None, 32, 32, 32)        128
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)        0
2D)

dropout_2 (Dropout)          (None, 16, 16, 32)        0

conv2d_4 (Conv2D)            (None, 16, 16, 64)        18496

batch_normalization_5 (Batc  (None, 16, 16, 64)        256
hNormalization)

conv2d_5 (Conv2D)            (None, 16, 16, 64)        36928

batch_normalization_6 (Batc  (None, 16, 16, 64)        256
hNormalization)

max_pooling2d_2 (MaxPooling  (None, 8, 8, 64)          0
2D)

dropout_3 (Dropout)          (None, 8, 8, 64)          0

flatten_1 (Flatten)          (None, 4096)              0

dense_2 (Dense)              (None, 128)               524416

batch_normalization_7 (Batc  (None, 128)               512
hNormalization)

dropout_4 (Dropout)          (None, 128)               0

dense_3 (Dense)              (None, 10)                1290

=================================================================
Total params: 592,554
Trainable params: 591,914
Non-trainable params: 640
```

شکل۱۷) شبکه تشکیل شده برای مدل ۲

بعد از تشکیل دادن شبکه هایمان نیاز به کامپایل کردن آنها داریم که برای اینکار از متد compile با اپتیمایزر adam استفاده می‌کنیم.

Compiling the model

```
[14]: def compile_model(model):
          """
          This function takes in the model returned from your get_model function, and compiles it with an optimiser,
          loss function and metric.
          Compile the model using the Adam optimiser (with default settings), the cross-entropy loss function and
          accuracy as the only metric.
          Your function doesn't need to return anything; the model will be compiled in-place.
          """
          model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

[15]: compile_model(model_1)
      compile_model(model_2)
      compile_model(model_3)
```

شکل۱۸) کامپایل کردن شبکه ها

بعد از کامپایل کردن شبکه به آموزش شبکه با استفاده از داده های اموزش می‌پردازیم که برای اینکار از متد fit استفاده می‌کنیم.

Training

```
[16]: def train_model(model, scaled_train_images, train_labels, batch_size, epochs):
          """
          This function should train the model for # epochs on the scaled_train_images and train_labels.
          Your function should return the training history, as returned by model.fit.
          """
          history = model.fit(x = scaled_train_images, y = train_labels, batch_size = batch_size, epochs = epochs)
          return history
```

Model 1 (1 VGG layer) ¶

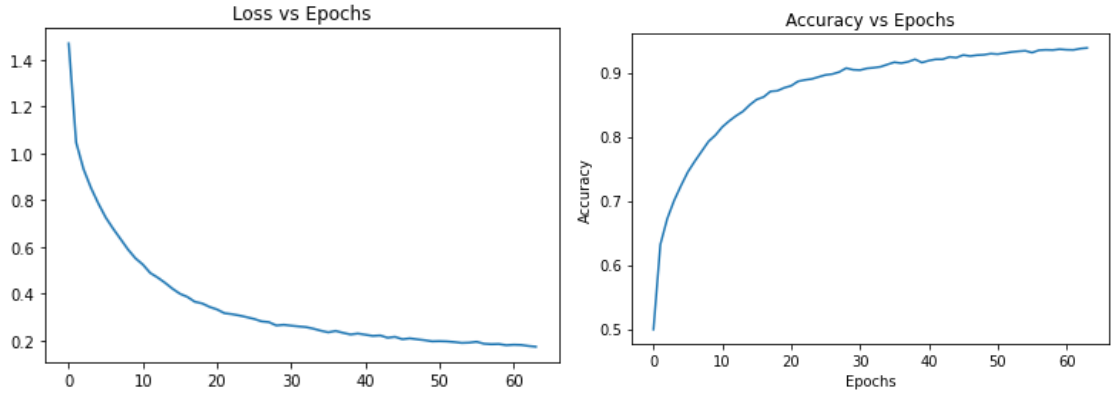
```
[17]: history_model_1 = train_model(model_1, x_train_s, y_train, batch_size=64, epochs=64)
```

شکل۱۹) اموزش شبکه ها

برای اموزش شبکه از بچ سایز های ۶۴تایی استفاده کردیم و به این علت که دوست عزیزمان! اقای طهرانی علاقه زیاده به دقت های بالا دارند از ۶۴ epoch استفاده کردند که ان‌شالله لپتاپشان بعد از ران گرفتن این شبکه ها سالم مانده باشد اما ما برای مقایسه دقت این ۳ مدل به همان ۱۰ epoch اکتفا می‌کنیم.

```
Epoch 60/64
782/782 [==============================] - 4s 6ms/step - loss: 0.1786 - accuracy: 0.9367
Epoch 61/64
782/782 [==============================] - 5s 6ms/step - loss: 0.1805 - accuracy: 0.9358
Epoch 62/64
782/782 [==============================] - 5s 6ms/step - loss: 0.1796 - accuracy: 0.9355
Epoch 63/64
782/782 [==============================] - 5s 6ms/step - loss: 0.1753 - accuracy: 0.9376
Epoch 64/64
782/782 [==============================] - 4s 6ms/step - loss: 0.1721 - accuracy: 0.9387
```

شکل۲۰) پایان آموزش شبکه ۱ با دقت ۹۳.۹ درصدی در ۶۴ epochs و دقت ۸۰.۲۵ درصدی در ۱۰ epochs



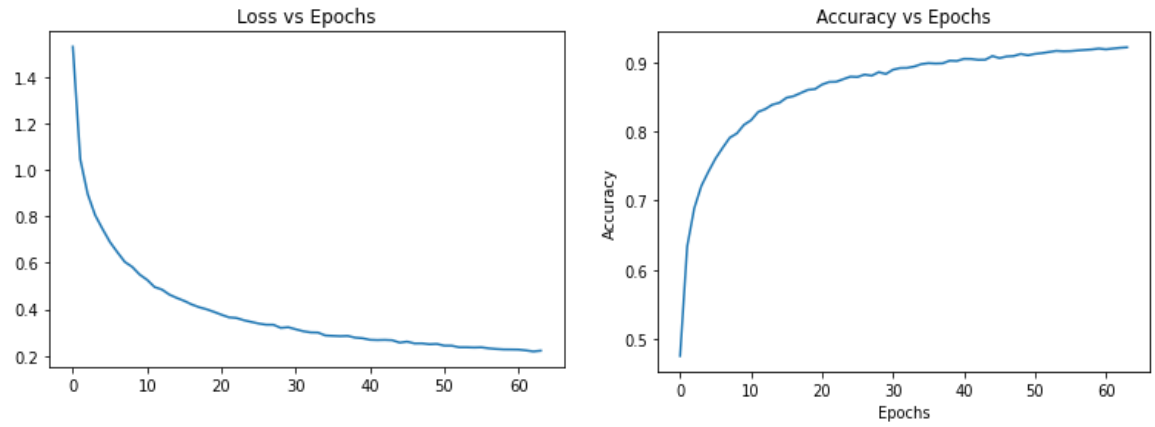
شکل۲۱) دقت و خطا شبکه ۱ بر حسب epochs

```
Model 2 (2 VGG layers)

[21]: history_model_2 = train_model(model_2, x_train_s, y_train, batch_size=64, epochs=64)
```

شکل۲۲) آموزش شبکه ۲

```
Epoch 60/64
782/782 [==============================] - 6s 8ms/step - loss: 0.2275 - accuracy: 0.9198
Epoch 61/64
782/782 [==============================] - 6s 8ms/step - loss: 0.2270 - accuracy: 0.9187
Epoch 62/64
782/782 [==============================] - 6s 8ms/step - loss: 0.2236 - accuracy: 0.9198
Epoch 63/64
782/782 [==============================] - 6s 8ms/step - loss: 0.2191 - accuracy: 0.9208
Epoch 64/64
782/782 [==============================] - 6s 8ms/step - loss: 0.2224 - accuracy: 0.9215
```

شکل ۲۳) پایان آموزش شبکه ۲ با دقت نهایی ۹۲.۱۵ درصدی و دقت ۸۰.۹۶ درصدی در epochs۱۰



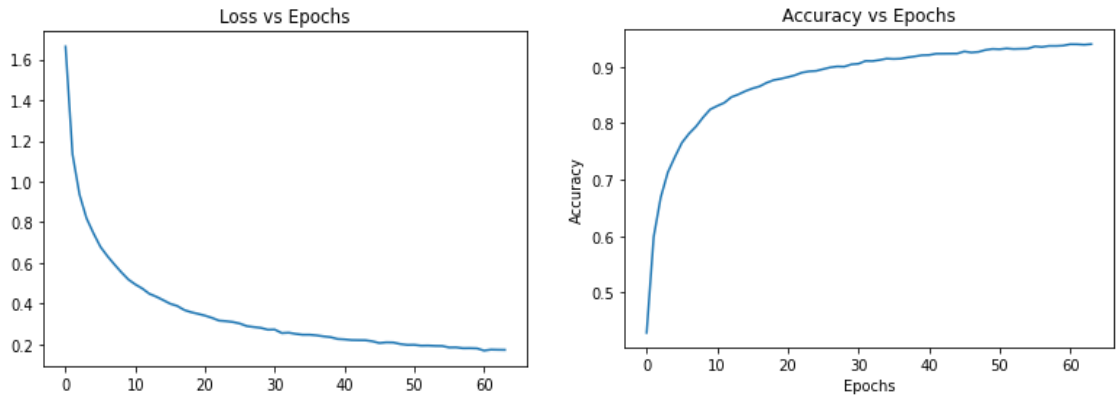
شکل۲۴) دقت و خطا مدل ۲ بر حسب epochs

```
Model 3 (3 VGG layers)

[25]: history_model_3 = train_model(model_3, x_train_s, y_train, batch_size=64, epochs=64)
```

شکل ۲۵) آموزش شبکه ۳

```
Epoch 60/64
782/782 [==============================] - 7s 9ms/step - loss: 0.1806 - accuracy: 0.9377
Epoch 61/64
782/782 [==============================] - 7s 9ms/step - loss: 0.1699 - accuracy: 0.9401
Epoch 62/64
782/782 [==============================] - 7s 9ms/step - loss: 0.1746 - accuracy: 0.9400
Epoch 63/64
782/782 [==============================] - 7s 9ms/step - loss: 0.1738 - accuracy: 0.9393
Epoch 64/64
782/782 [==============================] - 7s 9ms/step - loss: 0.1734 - accuracy: 0.9404
```

شکل ۲۶) پایان آموزش مدل۳ با دقت ۹۴ درصدی و دقت ۸۲.۴۵ درصدی در epochs۱۰



شکل۲۷) دقت و خطا مدل ۳ برحسب epochs

بعد از آموزش شبکه نوبت به تست دیتای تست بر روی هر ۳ مدل است که برای اینکار از متد evaluate استفاده می‌کنیم که در پایین نتیجه آن برای هر ۳ مدل را مشاهده می‌کنید.

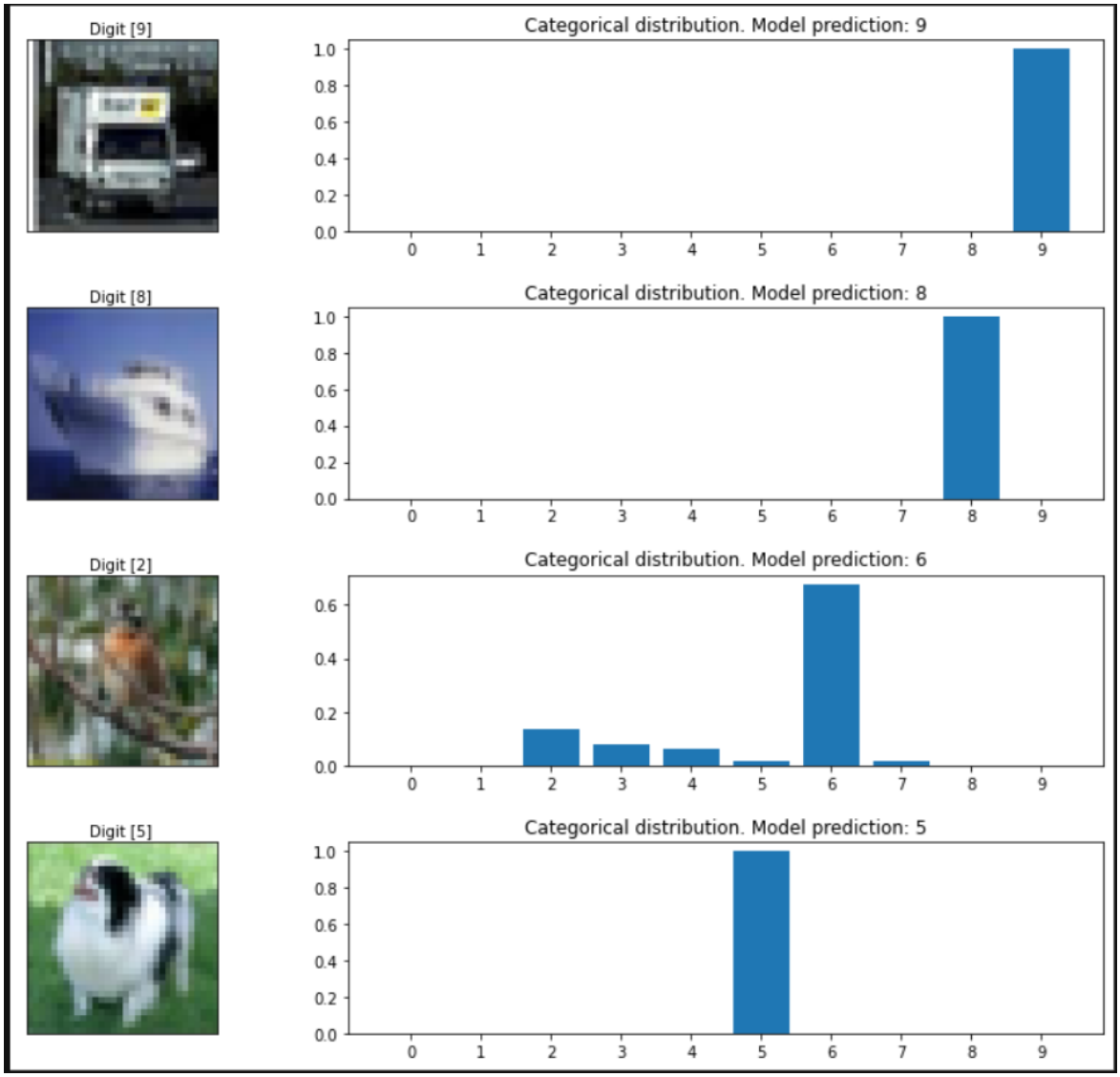
Model Evaluation

```
[29]: def evaluate_model(model, scaled_test_images, test_labels):
          """
          This function should evaluate the model on the scaled_test_images and test_labels.
          Your function should return a tuple (test_loss, test_accuracy).
          """
          history = model.evaluate(scaled_test_images, test_labels)
          return history
```

Model_1 (1 VGG layer)

```
[30]: test_loss, test_accuracy = evaluate_model(model_1, x_test_s, y_test)
      print(f"Test loss: {test_loss}")
      print(f"Test accuracy: {test_accuracy}")

      313/313 [==============================] - 1s 3ms/step - loss: 1.1251 - accuracy: 0.7252
      Test loss: 1.1250661611557007
      Test accuracy: 0.7251999974250793
```

شکل۲۸) تست شبکه که به دقت ۷۲.۵ درصدی ختم شد

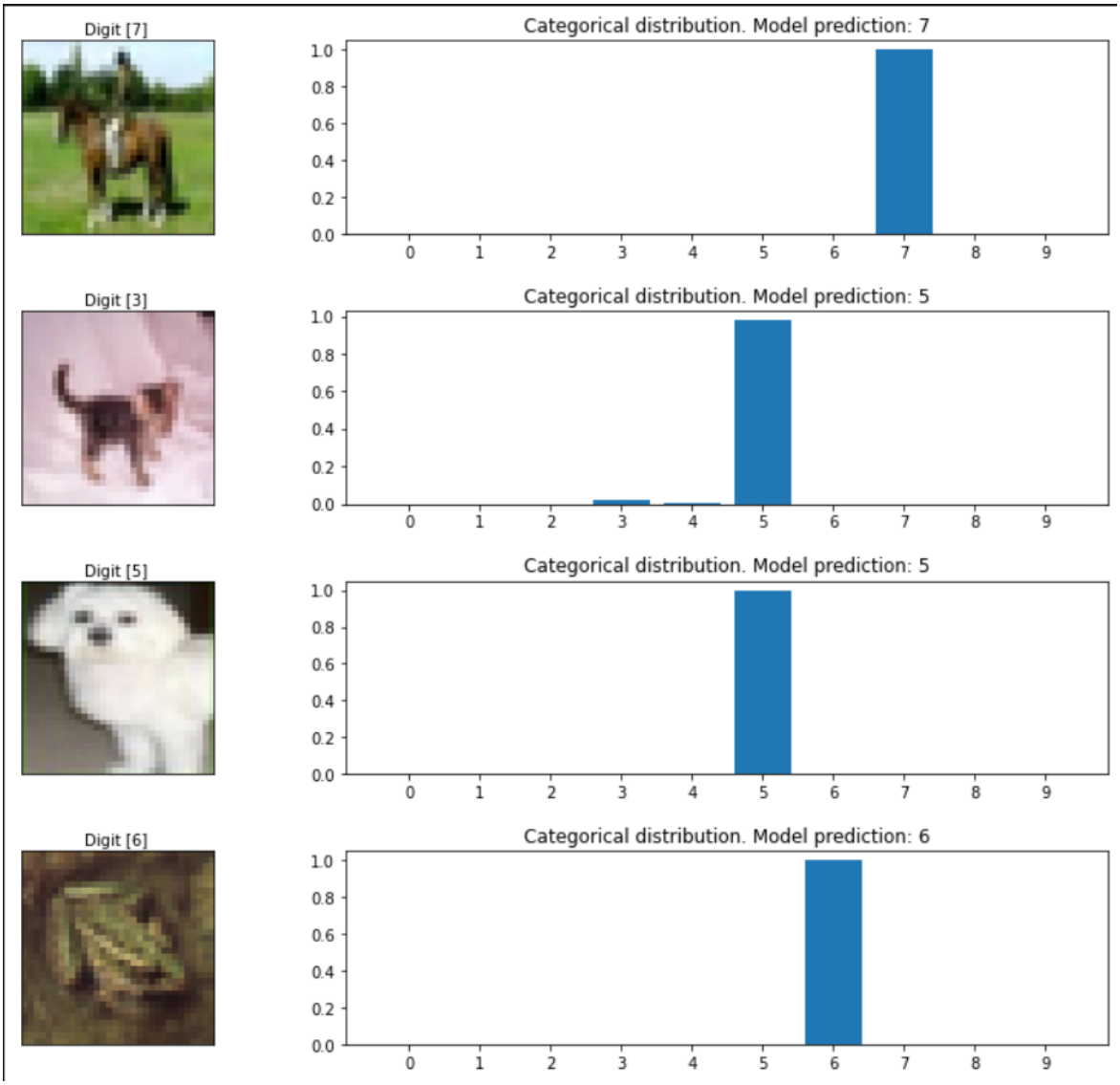


شکل۲۹) تست خروجی شبکه ۱

Model_2 (2 VGG layers)

```
[32]: test_loss, test_accuracy = evaluate_model(model_2, x_test_s, y_test)
      print(f"Test loss: {test_loss}")
      print(f"Test accuracy: {test_accuracy}")

      313/313 [==============================] - 1s 3ms/step - loss: 0.6241 - accuracy: 0.8233
      Test loss: 0.6240770220756531
      Test accuracy: 0.8233000040054321
```

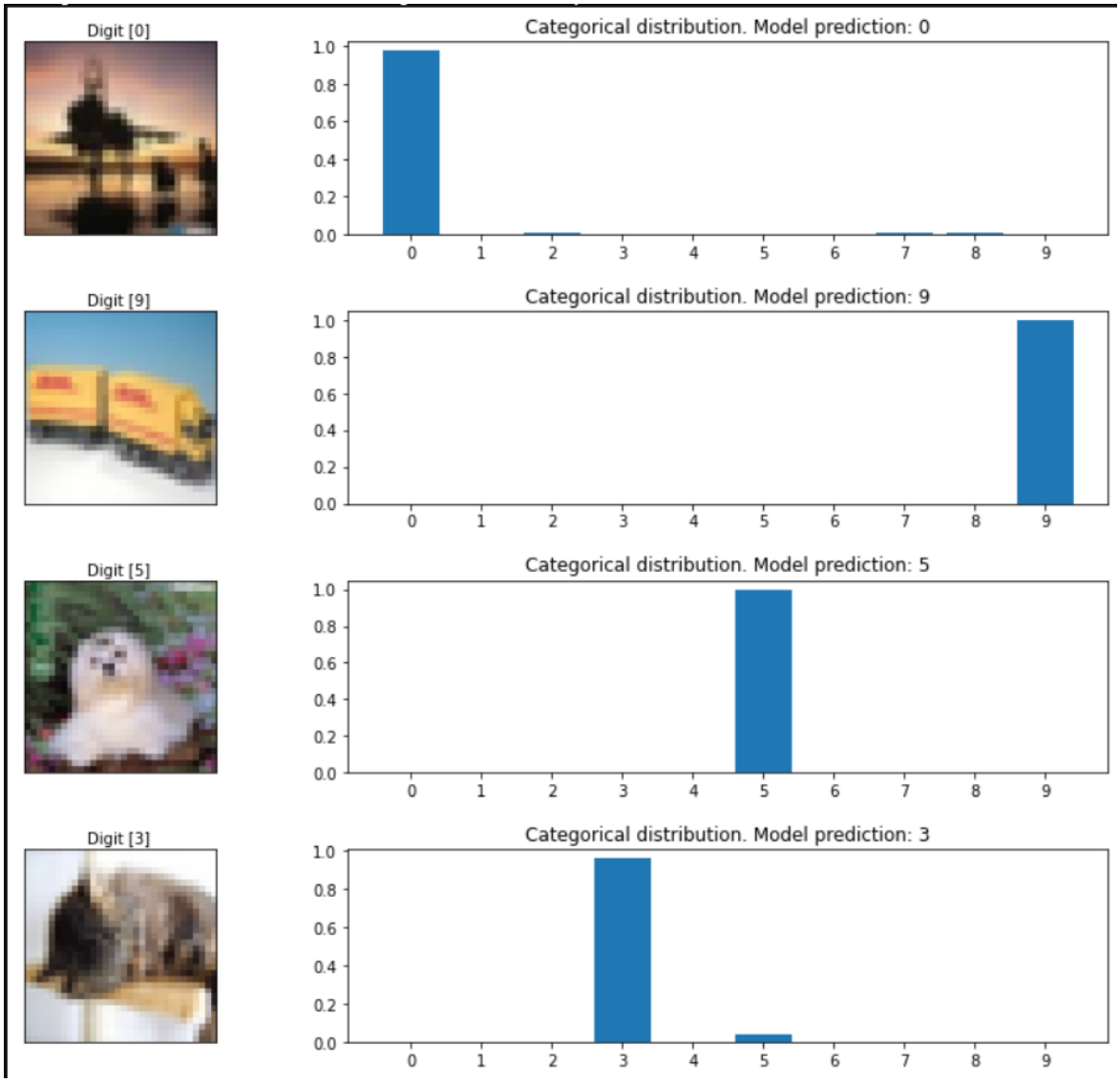
شکل۳۰) تست شبکه ۲ بر روی دیتا تست که دقت ۸۲.۳ درصدی منجر شد



شکل۳۱) خروجی شبکه ۲

```
Model_3 (3 VGG layers)

[34]: test_loss, test_accuracy = evaluate_model(model_3, x_test_s, y_test)
      print(f"Test loss: {test_loss}")
      print(f"Test accuracy: {test_accuracy}")

      313/313 [==============================] - 1s 4ms/step - loss: 0.4617 - accuracy: 0.8688
      Test loss: 0.4617026746273041
      Test accuracy: 0.8687999844551086
```

شکل۳۲) تست شبکه ۳ که دقت ۸۶.۸ درصدی منجر شد



شکل۳۳) خروجی شبکه ۳

همانطور که مشاهده شد با افزایش تعداد لایه ها دقت شبکه ها بیشتر شد که این مقدار تفاوت در شبکه اول و سوم تقریبا برابر با ۱۵ درصد است که مقدار قابل توجهی است و با توجه به اینکه در این ۳ مدل اختلاف میان دقت در داده های اموزش و تست قابل توجه نیست پس داده اموزش بر روی شبکه overfit نکرده است پس بهتر است از همان شبکه سوم استفاده کنیم.

پایان