

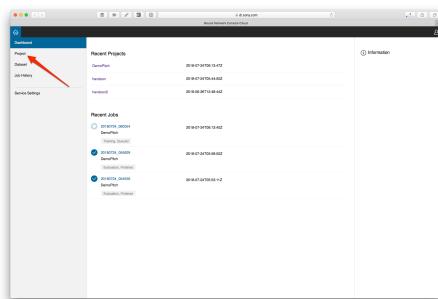
資料のURL <http://bit.ly/nnc-ho-1>

## 4と9を区別するディープラーニングを体験する

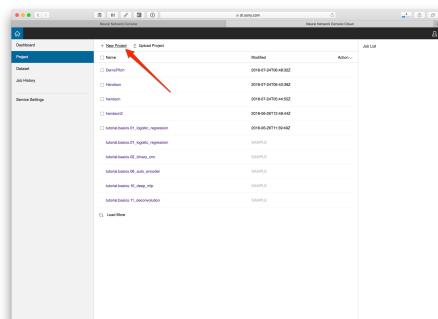
まず最初に28x28に書かれたモノクロの手書き画像を判別するディープラーニングを体験します。

### プロジェクトを作成する

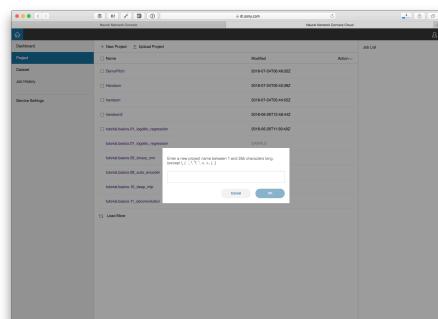
Neural Network Consoleにログインしたら、左側にあるプロジェクト（Project）を選択します。



New Projectを選択します。

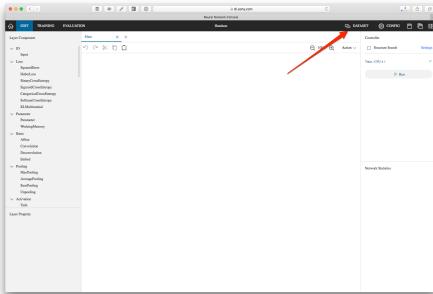


プロジェクト名はアルファベットや英数字が使えます。適当なプロジェクト名を（例えばHandsonなど）入力してOKボタンを押します。

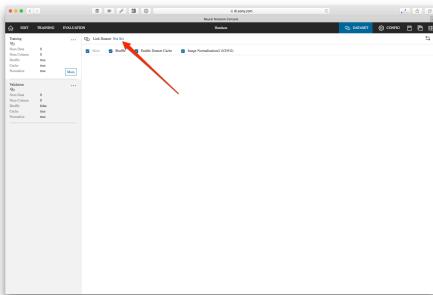


### データを読み込む

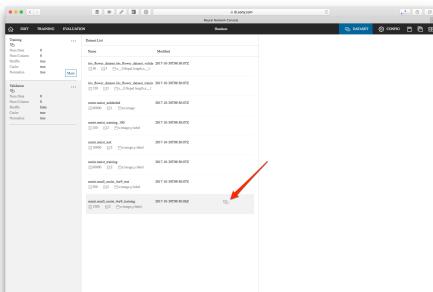
プロジェクトを作成すると、下のような画面が表示されます。まずデータを紐付けるために、右上にあるDATASETをクリックします。



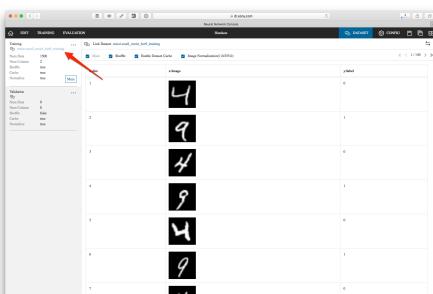
データはトレーニング (Training) と検証 (Validation) に分かれています。最初はトレーニングが選ばれている状態です。Not Setをクリックします。



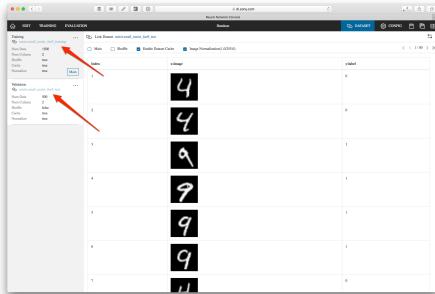
そうするとあらかじめ登録してあるデータが一覧表示されます。その中の mnist.small\_mnist\_4or9\_training にマウスを当て、右側にあるリンクアイコンをクリックします。



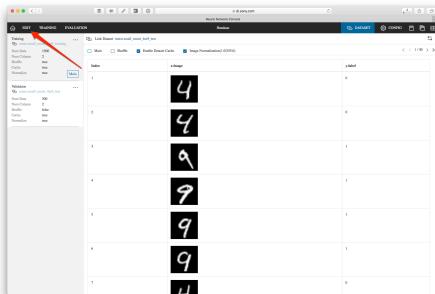
トレーニングのデータをリンクさせると、Trainingと書かれている欄の下にデータ名が表示されます。



同様に検証 (Validation) データとして mnist.small\_mnist\_4or9\_test を紐付けてください。

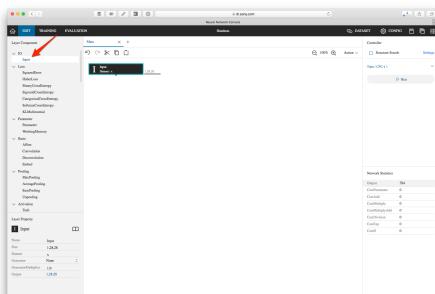


データの紐付けが終わったら、左上にあるEDITをクリックします。

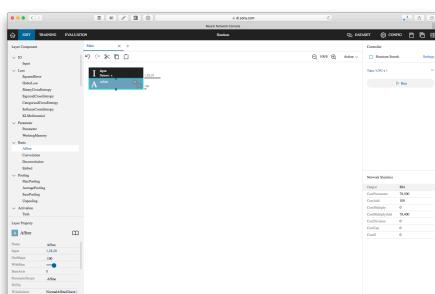


## アルゴリズムの設計

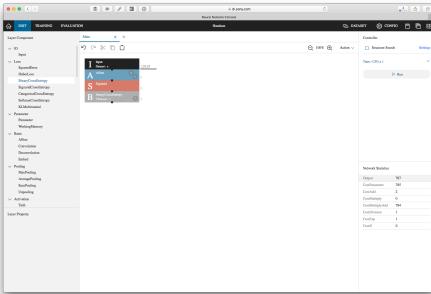
ではいよいよ機械学習のアルゴリズムを作っていきます。まず IO カテゴリにある Input をダブルクリックします（またはドラッグ&ドロップ）。これで入力データが追加されました。



続いてBasicにあるAffineを追加します。Affineで画像データの表示位置を調整します。このAffineを選択して、OutShapeを1にします。

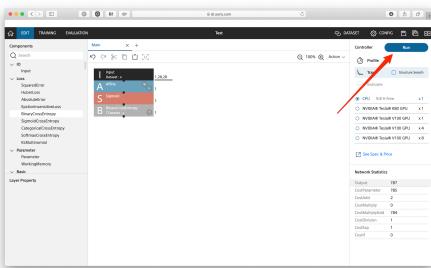


さらにActivation（活性化）としてSigmoidを追加します。これはシグモイド関数になります。最後に出力としてLossにあるBinaryCrossEntropyを追加します。

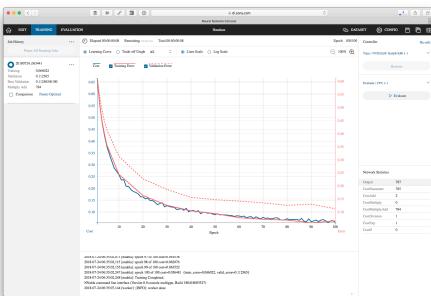


## トレーニングの開始

アルゴリズムの設計が終わったら、右側にある Run ボタンを押します。

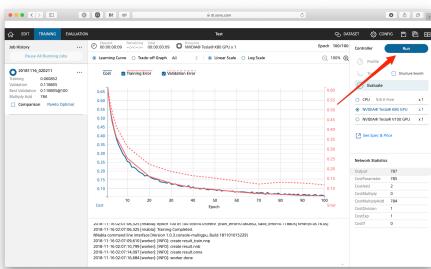


そうすると TRAINING タブに表示が移ってトレーニングが開始されます。CPUであったり、多人数で一氣に行うとキューが詰まってしまうかも知れません。その場合には終わるまでお待ちください。トレーニングが終わるとグラフが表示されます。トレーニングが収束しているのを確認してください。

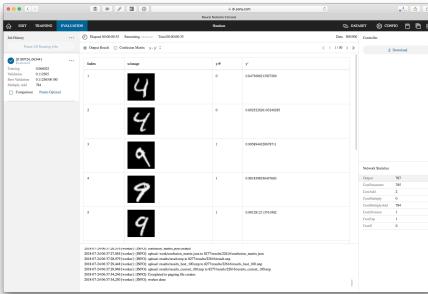


## 評価の開始

続いて右側にある Run ボタンを押します。

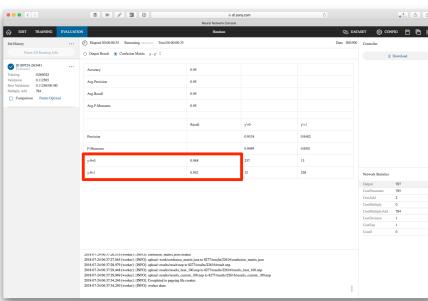


そうすると EVALUTE タブに表示が移って検証が行われます。処理が終わると画像の判定結果が表示されます。

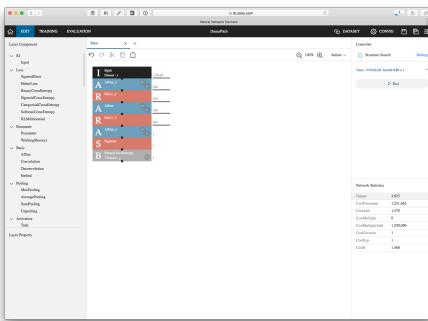


さらに上にあるConfusion Matrixを選ぶと処理全体の評価が確認できます。例えば今回は95%の精度で4と9を分類できたことが分かります。

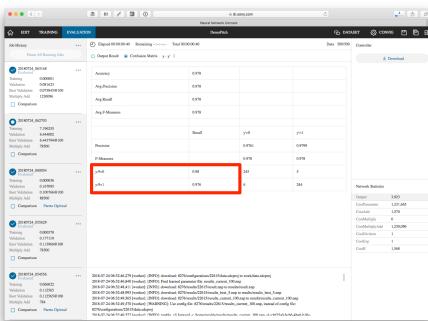
さらに詳細に書くと、今回は9ではない(9=0)つまり4の画像のうち、正しく4と判断できたものが94.8%、9の画像(9=1)のうち、9と正しく判断できたものが95.2%であったと表示されています。



現状のアルゴリズムに手を加えて再度トレーニング、評価してどう結果が変わるのが確認してみましょう。例えば以下のようなアルゴリズムを組んだとします。



この結果は約98%の精度で4と9を分類できています。



複数のトレーニングを行うことで、それぞれのグラフを比較できるようになります。TRAININGタブで、比較したいトレーニングデータのComparisonをチェックしてください。

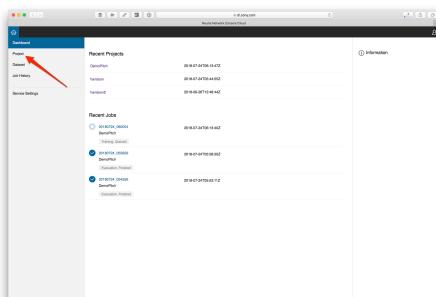
# あやめの種類を見分けるディープラーニングを体験する

次にあやめ (iris) の種類を見分けるディープラーニングを体験します。あやめは花びらの幅と長さ、がくの幅と長さの4つの要素によってIris setosa、Iris Versicolour、 Iris Virginicaの3つの種類に分類できます。

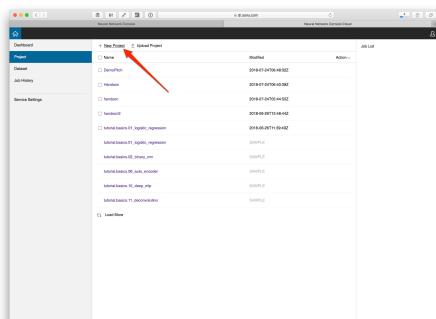
ぱっと見ただけでは殆ど区別がつきません。しかし上記のパラメータを用いることで、機械によって判別できるようになります。

## プロジェクトを作成する

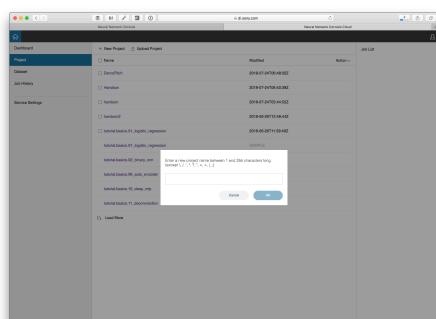
Neural Network Consoleにログインしたら、左側にあるプロジェクト (Project) を選択します。



New Projectを選択します。

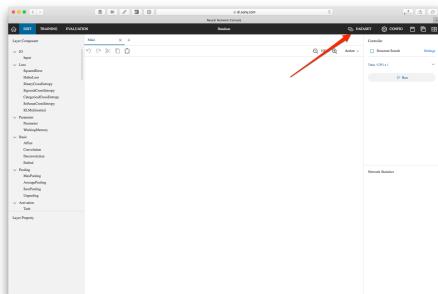


プロジェクト名はアルファベットや英数字が使えます。適当なプロジェクト名を (例えばHandsonなど) 入力してOKボタンを押します。

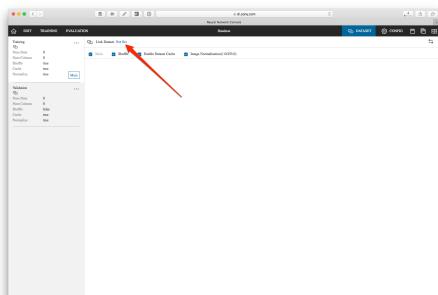


# データを読み込む

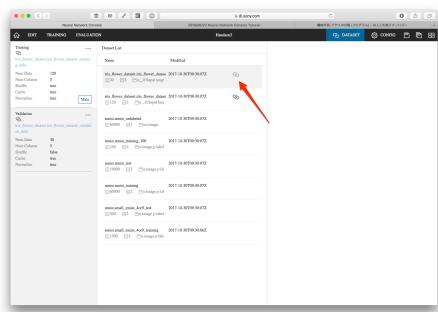
プロジェクトを作成すると、下のような画面が表示されます。まずデータを紐付けるために、右上にある DATASETをクリックします。



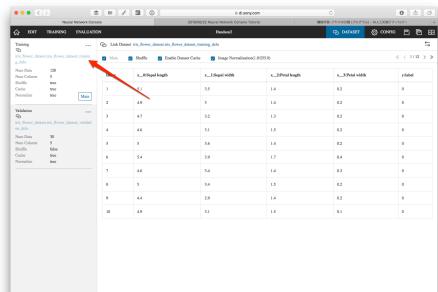
データはトレーニング（Training）と検証（Validation）に分かれています。最初はトレーニングが選ばれている状態です。Not Setをクリックします。



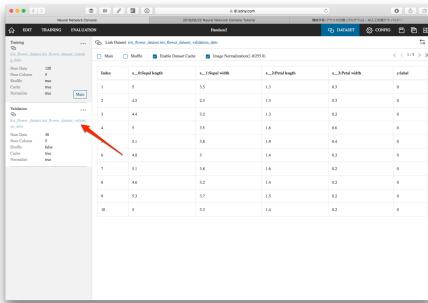
そうするとあらかじめ登録してあるデータが一覧表示されます。その中の `iris_flower_dataset.iris_flower_dataset_training_dele` にマウスを当て、右側にあるリンクアイコンをクリックします。



トレーニングのデータをリンクさせると、Trainingと書かれている欄の下にデータ名が表示されます。



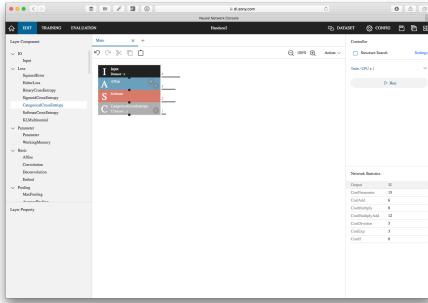
同様に検証 (Validation) データとして iris\_flower\_dataset.iris\_flower\_dataset\_validation\_del0 を紐付けてください。



The screenshot shows the 'DATA' tab of the Neural Network Designer. A red arrow points to the 'EDIT' button in the top-left corner of the main workspace. Below it, a table displays the Iris flower dataset with columns: Sepal Length, Sepal Width, Petal Length, Petal Width, and Class. The 'Validation' section at the bottom shows the validation set has been successfully linked.

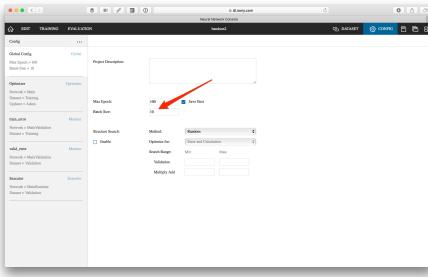
Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799	800	801	802	803	804	805	806	807	808	809	810	811	812	813	814	815	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895	896	897	898	899	900	901	902	903	904	905	906	907	908	909	910	911	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991	992	993	994	995	996	997	998	999	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	13

さらにActivation（活性化）としてSoftmaxを追加します。これはソフトマックス関数になります。ソフトマックス関数によって、結果を確立に変換します。最後に出力としてLossにあるCategoricalCrossEntropyを追加します。



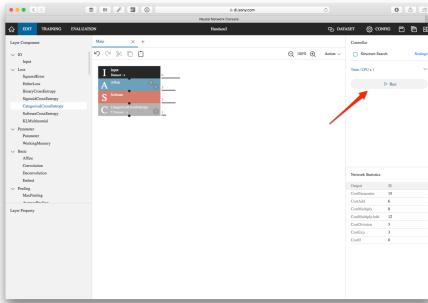
## CONFIGの変更

右上にあるCONFIGをクリックしてCONFIGを表示します。Batch Sizeを10とします。

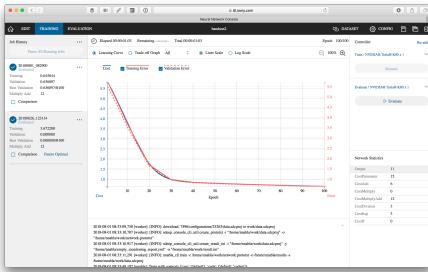


## トレーニングの開始

アルゴリズムの設計やCONFIGの変更が終わったら、EDITタブに戻って右側にあるRunボタンを押します。

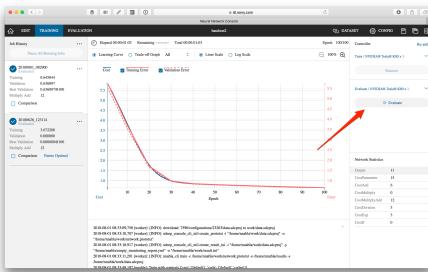


そうすると TRAINING タブに表示が移ってトレーニングが開始されます。CPUであったり、多人数で一氣に行うとキューが詰まってしまうかも知れません。その場合には終わるまでお待ちください。トレーニングが終わるとグラフが表示されます。トレーニングが収束しているのを確認してください。



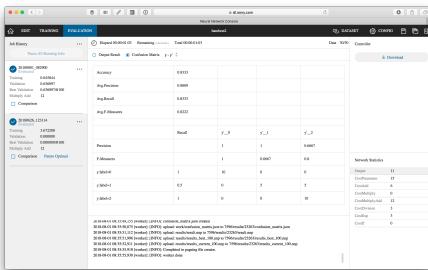
## 評価の開始

続いて右側にある Run ボタンを押します。



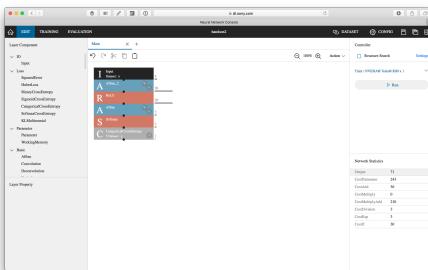
そうすると EVALUTE タブに表示が移って検証が行われます。処理が終わると評価結果が表示されます。評価結果を見ると、次のようなことが分かります。

- **Iris Setosa (y=0)** → 100% (10/10正しく評価)
- **Iris Versicolour (y=1)** → 50% (10あるデータの内、5つしか正しく判定できていない)
- **Iris Virginica (y=2)** → 66% (10あるデータは10正しく判定、ただし y=1のデータが5つ誤判定)



## アルゴリズムの更新

現状のアルゴリズムに手を加えて再度トレーニング、評価してどう結果が変わるのが確認してみましょう。例えば以下のようなアルゴリズムを組んだとします。



この結果は100%正確に分類分けできます。

The screenshot shows the TensorFlow Model Zoo interface. On the left, there's a sidebar with 'All Models' and 'Training' sections. Under 'Training', two runs are listed: '20180514\_00114' and '20180514\_01204'. The right side has tabs for 'Training', 'Evaluation Metrics', and 'Comparison'. The 'Comparison' tab is active, showing a table with columns 'Model', 'F1-0', 'F1-1', and 'F1-2'. The first row (20180514\_00114) has values 1, 1, and 1 respectively. The second row (20180514\_01204) has values 0.99, 0.99, and 0.99. Below the table is a 'Network Statistics' section with various metrics like 'Epochs' (7), 'Confidence' (94%), and 'Loss' (0.0001).

複数のトレーニングを行うことで、それぞれのグラフを比較できるようになります。TRAINING タブで、比較したいトレーニングデータの Comparison をチェックしてください。

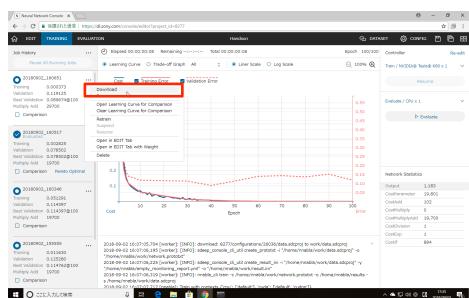
# 作成したモデル、ネットワークをローカルコンピュータで体験する

では作成した4か9かを見分けるディープラーニングをローカルコンピュータで使ってみましょう。CLI（コマンドライン プログラミング インタフェース）であれば今回はプログラミングを書かずに試す方法を紹介します。なお、今回はクラウド版を対象に紹介します。

## ネットワークと学習済みのパラメータをダウンロードする

まず作成したネットワークと学習済みのパラメータをダウンロードします。これはトレーニング (TRAINING) または検証 (EVALUTION) タブに移動して、ジョブ履歴の中からダウンロードしたい学習結果の三点リーダー (...) をクリックします。そして出てきたメニューのDownload > NNP(Neural Network Libraries file format)をクリックします。

最初の課題である4/9を判別するディープラーニングのネットワークをダウンロードしてください。アヤメではありません。



そうすると result\_train.nnp というファイルがダウンロードされます。これがディープラーニングのネットワークと学習済みのパラメータが入った内容になります。

ここからPython環境を整えますが、Google Colaboratoryを使う場合、Pythonをインストールする必要はありません。ただし、Webアプリケーションを試すことはできませんので注意してください。

=====

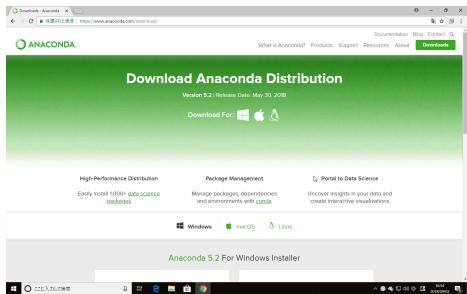
ここからはPythonをローカルにインストールする方向け

## Google Colaboratoryは後述しますのでスキップしてください

## Python環境を整える

このファイルを扱うためにPython 3.6をインストールします。ディープラーニングを行っていくのであれ

ば素のPythonよりも[Anaconda](#)をインストールする方が良いでしょう。Anacondaはディープラーニングを行うのに便利な、必須とも言えるライブラリがあらかじめインストールされています。macOSやLinuxについてもデフォルトでは3.7系のPythonになってしまい、nnablaがインストールできなかったり、算術系のライブラリが不足しているのでAnacondaの利用をお勧めします。



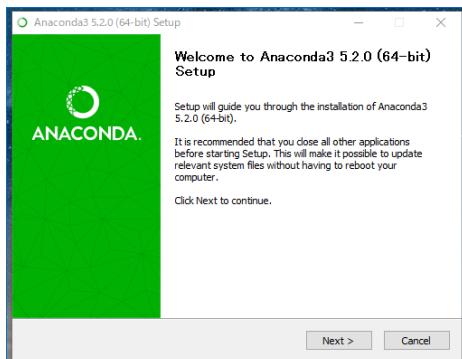
AnacondaはWindows/macOS/Linux向けにそれぞれリリースされています。自分の環境に合わせたものをダウンロード、インストールしてください。

## [Downloads - Anaconda](#)

## Anacondaを実行する

Windowsの場合は、Anacondaをスタートメニューから実行します。そうするとパスにPythonなどが入った状態のコマンドプロンプトが立ち上ります。macOSやLinuxの場合はターミナルを立ち上げて、以下のコマンドを入力するとAnacondaのPythonが優先実行されます。

```
export PATH="/anaconda3/bin:$PATH"
```



まずPythonのライブラリ管理であるpipをアップデートします。

```
python -m pip install --upgrade pip
```

次にディープラーニング用ライブラリのnnablaをインストールします。

```
pip install nnabla
```

nnablaをインストールすると、以下のコマンドが使えるようになります。

```
nnabla_cli
```

これがNNCを試す際のコマンドになります。これで準備完了です。

=====

ここまでPythonをローカルにインストールする方向け

**「デモをダウンロードする」までスキップしてください。**

## Google Colaboratoryで環境を整える

[Google Colaboratory](#)にアクセスして「新しいPython3のノートブック」を作成します。

ノートブックが開いたら、まずディープラーニング用ライブラリのnnablaをインストールします。!はGoogle Colaboratoryでコマンドを実行する際に付与するプリフィックスです。

```
!pip install nnabla
```

記述した後、Shift + Enterで実行されます。

nnablaをインストールすると、以下のコマンドが使えるようになります。

```
!nnabla_cli
```

これがNNCを試す際のコマンドになります。これで準備完了です。

## デモをダウンロードする

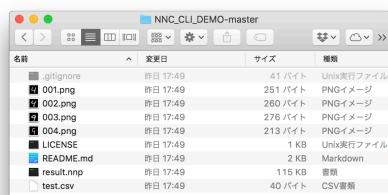
デモ用のリポジトリを用意しています。こちらをダウンロードして、Zipを展開してください。

ローカル実行の場合

```
http://bit.ly/nnc_cli_demo
```

Google Colaboratoryの場合

```
!wget https://github.com/goofmint/NNC_CLI_DEMO/archive/master.zip  
!unzip master.zip  
!cd NNC_CLI_DEMO-master
```



中には以下のようなファイルがあります。

- 001~004.png  
テスト用の画像
- result.nnp  
あらかじめダウンロードしておいたNNCのモデル/ネットワーク
- test.csv  
テスト用のCSVファイル
- web/  
[ハンズオン4回目](#)の内容です

このディレクトリに、あなたがダウンロードした (result\_train.nnp) を置いてください。

## Google Colaboratoryの場合

Google Colaboratoryの場合は、以下を実行するとファイル選択ダイアログが表示されます。時間制限があるようですが、表示した後すぐにresult\_train.nnpを選択します。

```
from google.colab import files
uploaded = files.upload()
```

## test.csvについて

test.csvは以下のような内容になっています。今回は検証目的に使うので、yカラムはありません。

```
x:image
001.png
002.png
003.png
004.png
```

指定されている画像は、同じフォルダ内にある画像ファイルを指定しています。

## 実行する

では実際に試してみましょう。上記のファイルをダウンロード、解凍したディレクトリでコマンドを以下のように実行します。

□一カルの場合

```
nnabla_cli forward -c result_train.nnp -d test.csv -o ./
```

Google Colaboratoryの場合

```
!nnabla_cli forward -c result_train.nnp -d test.csv -o ./
```

-c で学習した内容を、 -d で評価するCSVファイルを指定します。最後の -o は実行結果の出力先です。

実行すると `output_result.csv` と `progress.txt` が出力されます。大事なのは `output_result.csv` です。内容を見ると、以下のようになっています（実行環境によって異なるかも知れません）。

```
x:image,y'  
/path/to/001.png,0.00019709873595274985  
/path/to/002.png,1.0769620750750164e-08  
/path/to/003.png,0.9999932050704956  
/path/to/004.png,0.9113416075706482
```

今回は 001~002.pngまでが4を書いたもの、003~004.pngが9を書いたものになります。ただしく判定されているのが分かります。

Google Colaboratoryの場合は以下で結果を確認できます。

```
!cat output_result.csv
```

## 別な画像ファイルで試す

画像は[myleott/mnist\\_png](#)にあるものをお借りしています。他にもたくさんの4、9画像がありますので、ダウンロードして試してみてください。ファイルを追加したらtest.csvを更新するのを忘れないでください。

[http://bit.ly/mnist\\_png](http://bit.ly/mnist_png)

このように自分で作ったモデル、ネットワークをコンピュータ上で実行できます。コマンドを実行して、結果のCSVファイルを読むという方法であればPython以外のプログラムからでもNNCが利用できます。

# 作成したモデル、ネットワークをWebアプリケーションで体験する

3回目のCLIで作った環境をそのまま使っていきます。もしAnacondaやnnablaをインストールしていなかったら、[ハンズオン3回目](#)の内容を実施してください。

今回はPythonを使って、Webアプリケーションの中にどう組み込めば良いかを解説します。

## Webフレームワークをインストールする

今回はPython製の簡易的なWebフレームワーク `bottle` を使います。これはコマンドプロンプトやターミナルで実行します。

```
pip install bottle
```

## HTMLの解説

まずWebアプリケーションの画面を紹介します。今回はCanvasタグを使ってマウスで文字を書き、それをサーバに送る仕組みとします。JavaScriptのコードはコメントを参照してください。コードは[endoyuta/mnist\\_test](#)のHTMLを参考にさせてもらっています。ファイルは前回ダウンロードしたGitリポジトリのwebフォルダの中になります。

## app.pyの編集

編集前は `web/app.py` は以下のようになっています。

```
def image():
    img_str = request.params['img']
    if img_str == False:
        return "{}"

    # 画像を28x28、グレースケールに変換
    b64_str = img_str.split(',')[1]
    img = Image.open(BytesIO(a2b_base64(b64_str)))
    img = img.resize((28, 28))
    img = img.convert('L')

    # Neural Network Consoleのファイルを読み込む

    # 実行するネットワークを指定する

    # 入力変数を取得

    # 出力変数を取得

    # 画像を配列に変換、数値を0～1.0にする

    # 判定する

    # 結果を出力する
```

コード中のコメントアウトされている部分を編集していきます。

## Neural Network Consoleのファイルを読み込む

ダウンロードした result.nnp を読み込みます。 `web` フォルダの一つ上の階層にあるはずなので、以下のように記述します。大文字、小文字も区別されるので注意してください。

```
# Neural Network Consoleのファイルを読み込む
nnp = nnp_graph.NnpLoader('../result_train.nnp')
```

## 実行するネットワークを指定する

ネットワーク/モデルを読み込んだら、実行するネットワークを取得します。`MainRuntime` という名前は Neural Network ConsoleのCONFIGタブで確認できます。

```
# 実行するネットワークを指定する
graph = nnp.get_network('MainRuntime', batch_size=1)
```

## 入力変数を取得

入力変数（今回は`x`）を取得します。入力が複数ある場合は、複数指定しなければいけません。今回は一つ

なので、入力値の最初の値（一つしかないので）を指定します。

```
# 入力変数の名前を取得  
input = list(graph.inputs.keys())[0]  
x = graph.inputs[input]
```

## 出力変数の名前を取得

出力も同じように取得します。今回は一つしかないので入力と同じように取得できます。

```
# 出力変数の名前を取得  
output = list(graph.outputs.keys())[0]  
y = graph.outputs[output]
```

## 画像を配列に変換、数値を0～1.0にする

HTMLから受け取った画像を配列に変換する際には numpy というライブラリが使えます。ディープラーニングを行う際にはよく使うライブラリです。そして、NNCでは-1.0～1.0の値のみ扱えますので、配列を255で割ります。

```
# 画像を配列に変換、数値を0～1.0にする  
x.d = np.array(img) * (1.0 / 255.0)
```

## 判定する

入力変数の準備ができたら、後は判定するだけです。判定は以下のようにします。

```
# 判定する  
y.forward(clear_buffer=True)
```

## 結果を出力する

結果をHTMLに返します。この時、HTMLでも判別しやすいようにJSON形式で返します。結果は `y.d` の中に入っています、今回の場合は `y.d[0][0]` で取得できます。

```
# 結果を出力する  
response.content_type = 'application/json'  
return '{{"result": {}}}'.format(y.d[0][0])
```

## 実行する

ではPythonを実行します。

```
cd web  
python app.py
```

その後、Webブラウザで <http://localhost:8080/> にアクセスします。HTML画面が表示されれば成功です。



手書きで4または9を書いて、サーバに送信します。ディープラーニングによって判定され、その結果が返ってきます。



今回のコードで、ディープラーニングのネットワークを作るコードは一切書いていないのが分かります。ダウンロードした `result.nnp` ファイルからディープラーニングのネットワーク情報や重みを取得することで、プログラミングはとてもシンプルになります。

# より複雑なネットワークをWebアプリケーションで試す

今回はより複雑なネットワークを体験します。4か9だけでなく、0～9までの数値を判別できるようにします。

## 作成するネットワーク

---

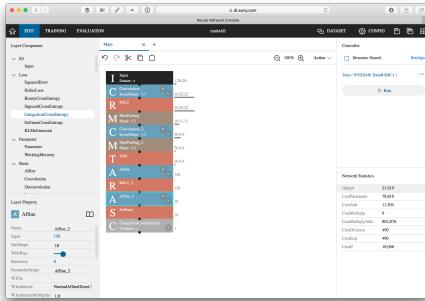
### データセット

新しいプロジェクトを作成して、`mnist.mnist_training` をトレーニング、`mnist.mnist_test` をテストデータとして指定してください。

### ネットワーク

作成するネットワークは以下のようになります。パラメータを変更しているものについては下に設定を記述しています。

- **Input**
- **Convolution**  
KernelShape : 7,7  
WithBias : True
- **ReLU**
- **MaxPooling**  
IgnoreBorder : True
- **Convolution**  
OutMaps : 30  
KernelShape : 3,3
- **MaxPooling**  
IgnoreBorder : True
- **Tanh**
- **Affine**  
OutShape : 150
- **ReLU**
- **Affine**  
OutShape : 10
- **Softmax**
- **CategoricalCrossEntropy**

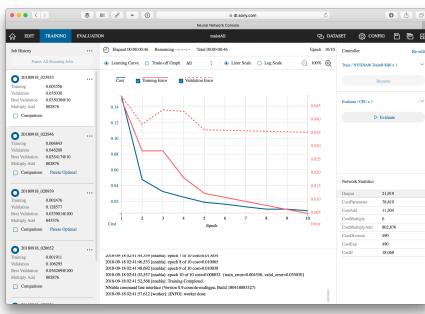


## 設定

CONFIGで、Max Epochを10にします。

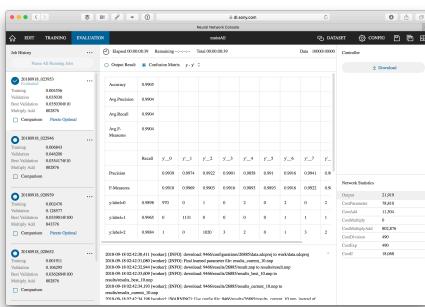
## 学習の実施

トレーニングを実施します。10エポックなのですぐに終わるでしょう。



## 検証の実施

続けて検証を実施します。98%くらいの精度になるでしょう。



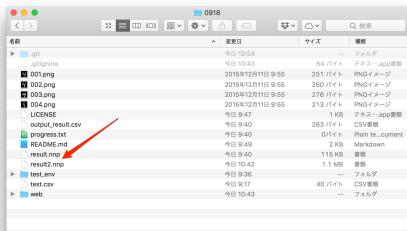
## ネットワークのダウンロード

検証が終わったら、そのネットワーク情報をダウンロードします。

The screenshot shows the NNC interface with the 'EVALUATION' tab selected. It displays a table of job history with columns for Job ID, Status, Training, Validation, Best Validation, and Multiply Add. A context menu is open over the first job entry, listing options like 'Download', 'Rename', 'Retrain', 'Suspend', 'Resume', 'Open in EDIT Tab', and 'Delete'. A red arrow points from the text '右クリックしてメニューを開く' to the 'Delete' option in the menu.

Job ID	Status	Training	Validation	Best Validation	Multiply Add
20180918_023953	Evaluated	0.004556	0.035030	0.035030@10	802876
20180918_022946	Pending	0.006843	0.046200	0.035417@10	802876

ダウンロードしたら result2.nnp としてCLIで実行したresult.nnpと同じフォルダにいれます。あらかじめ result2.nnp は存在しますが、上書きして構いません。



## web/app.pyの修正

今回は 0または1ではなく、0~9それぞれの数字において可能性が返ってきます。イメージとしては書きのようになります。最初が0の可能性、次が1の可能性...となります。この中で一番1に近いものが対象の数値になります。今回であれば8です。

```
[ 3.5272753e-03 1.0275582e-07 3.2790832e-03 3.3631185e-03 3.5058970e-06
 1.4182813e-03 2.3267221e-05 1.2304156e-06 9.7950816e-01 8.8759838e-03 ]
```

e-01は浮動小数点表記になります。9.7e-01は0.97、8.8e-03は0.0088になります。通常は一番eが少ないものが対象になるでしょう。

これらは可能性なので、100で割るとパーセントになるものです。つまり最も1.0近いものが手書きされた文字になります。

## 読み込むネットワークの変更

先ほどダウンロード、配置したnnpファイルに変更します。

```
# Neural Network Consoleのファイルを読み込む
# 修正前
nnp = nnp_graph.NnpLoader('../result_train.nnp')

# 修正後
nnp = nnp_graph.NnpLoader('../result_train-2.nnp')
```

## 関数の追加

0～9の可能性について、最も1に近いものを算出する関数を追加します。

`run(host='localhost', port=8080)` の上に追加します。

```
# この関数を追加
def getNearestValue(list, num):
    # リストの差分を絶対値で算出し、差分が一番小さいインデックスを取得
    idx = np.abs(np.asarray(list) - num).argmin()
    return idx

run(host='localhost', port=8080)
```

via [Pythonのリスト要素からある値と最も近い値を取り出す - Qiita](#)

## データを返す部分を修正

画像を判定した後、先ほど追加した関数を実行して最も1に近い可能性を持った数値を取得します。そして、それをHTMLに返します。

```
# HTMLに返す
number = getNearestValue(y.d[0], 1.0)      # ← 追加
return '{{"result": {}}}'.format(number)    # ← 修正
```

## web/index.html を修正

前はPythonからの返却値に応じて処理を判別していましたが、それが不要になります。一番下の方にある表示処理を修正します。

```
// 修正前
document.querySelector("#answer").innerText = (json.result > 0.2 ? '9' : '4') + 'です'

// 修正後
document.querySelector("#answer").innerText = json.result + 'です';
```

これで修正は完了です。修正後のファイル（web/app.py）を以下にアップロードしておりますので、問題があったら確認してください。

[NNCデモ（0～9の数字判定）](#)

## 実行する

ではPythonを実行します。

```
cd web  
python app.py
```

その後、Webブラウザで <http://localhost:8080/> にアクセスします。HTML画面が表示されれば成功です。



手書きで0～9の数字を書いて判定させてみましょう。正しい判定が行われていれば、ネットワークやモデルが正しく組めているということになります。



今回のデモで分かる通り、NNCの処理自体は殆ど変更していません。nnpファイルにはネットワーク構造や重みに関する情報が入っていますので、それらを用いることで再利用性の高いコーディングが実現できます。